ECE4191 Final Report

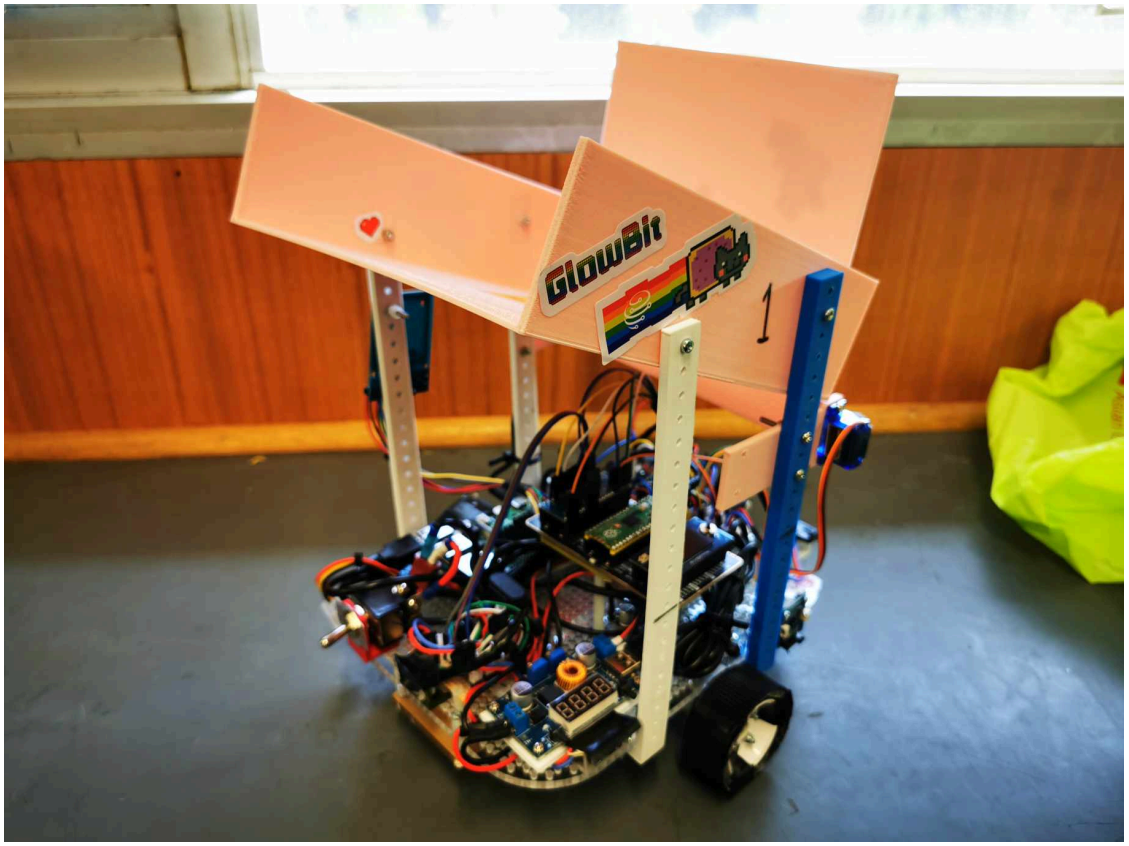


Figure 1: Robot Full Assembly

Team Number/Name:

Team 1

Team Member Student Numbers:

Student Name
Terence Ting
Yide Tao
Luqi Huang
Jia Ming Ong
Ayrton Zuglian

Executive Summary

The goal of the project is to develop a robot capable of delivering parcels to predetermined locations around an arena while avoiding collision with, and potentially collaborating with, other robots in the same arena [1]. The project team consists of five members, each responsible for the development and integration of some subsystem of the overall robot.

The team has decided on a metric-based approach for robot navigation. A virtual 2D map of a pre-constructed arena can be preloaded into the onboard computer of the robot, to provide the system with location information and dynamic modelling capabilities.

The final robot is an integrated product of several subsystems, consisting of a power supply unit to power the robot, a sensor suite for localisation, a navigation system for the robot to successfully operate in the arena, mechanical systems to interact with and deliver the parcel, and a communications protocol to collaborate, as well as prevent collisions with another robot.

The capabilities of the robot shown in this report will be demonstrated in a competition arena, where two robots will cooperate with each other to deliver parcels within a given time. Strategies for optimising both the actions of a single robot, and protocols for collaboration between the two cooperating robots have been developed to ensure successful completion of the tasks.

Contents

1.0 Introduction

The objective of this project is to develop a parcel delivery solution for warehouse operations in the form of a delivery robot. This robot is expected to cost around $200, and be able to identify and deliver three different parcel types to their respective locations.

The dimensions of the warehouse in which delivery will occur is 1.2m x 1.2m, and each robot will operate simultaneously with another robot, so a navigation and object avoidance system will be essential to ensure operational safety in such an enclosed space.

Overall, the design of this robot leans heavily on these criteria, attempting to balance cost, reliability, efficiency and ease of use and caters to both internal and external stakeholders.

1. **Internal Stakeholders:**

    - **Modularised Design:** The design team should incorporate relevant system engineering protocols in the design process.

    - **Supply Security:** The design team needs to consider the availability for key components in the market (e.g. Raspberry PI).

    - **Clean Appearance:** The robot should also have a relatively clean appearance. The design team should also follow the rules set out in the prototyping guide.

2. **External Stakeholders:**

    - **Cheap:** Common warehouses would require a large number of robots to maintain operational efficiency, the customers are generally very sensitive to the unit price for each robot, it is really important to keep the price of the robot low.

    - **Robust:** Previous designs offer highly reliable systems, capable of achieving 99% operation accuracy [1] in inventory management, similar Performance is expected for our design.

    - **Collaboration:** The delivery solution is expected to be able to collaborate with warehouse robots from other development teams.

    - **Follow the Environment and safety regulations:** The system should consider common safety requirements such as AS4024, ANSI/ITSDF B56.5-2010 and sustainability design goals outlined by the UN.

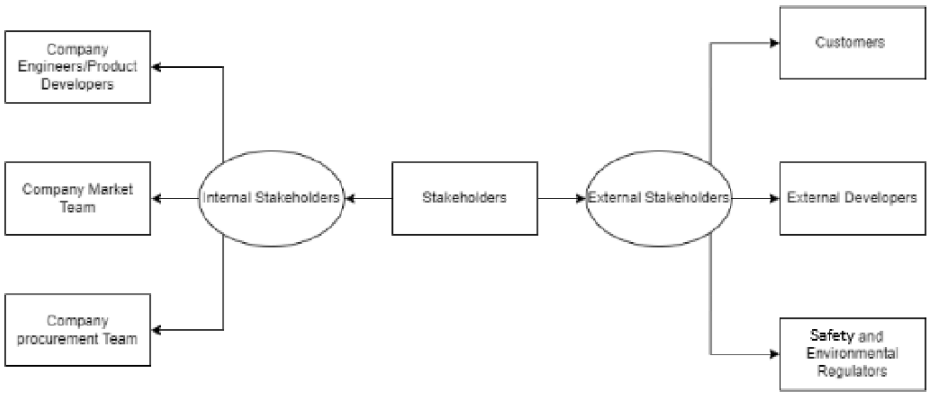


Figure 1.1 - Visualisation of Project Stakeholders

2.0 Problem Statement

The team has been tasked with designing a robot to deliver three different types of parcels to their respective locations, working together with another robot in a 1.2m x 1.2m arena. The robot must be able to identify when a parcel has been placed on it, identify which type of parcel has been placed and decide which stationary bin said parcel is to be delivered. Using its delivery mechanism, the robot must then successfully deliver the correct parcel to the correct bin. Then, it must return to the loading bay in order to have another parcel loaded and repeat the process. The robot must be reliable, efficient, cheap and able to communicate with other robots with dissimilar processors, delivery systems and navigation algorithms. During operation, the robot must also cooperate with another robot with the same goal, that is also operating within the same arena.

2.1 User Requirements

ID	Description	Classification	Analysis
1	**ECE4191 Robot**	Heading	Not Applicable
2	**Navigation**	Heading	Not Applicable
3	The robot will be operating in a warehouse with the dimensions of 1.2mx1.2m	Information	Not Applicable
4	The robot shall be able to navigate from a point of origin to three different waypoints	Requirement	Accepted
5	The robot shall be able to detect obstacles	Requirement	Obsolete

6	The robot shall be able to make a path correction to avoid collision with an obstacle	Requirement	Obsolete
7	The robot shall move continuously with pauses or sharp movements	Requirement	Accepted
8	The robot shall be capable of communicating with other robots to avoid collisions	Constraint	Accepted
9	The robot shall be capable of returning to its point of origin after arriving at a destination	Requirement	Accepted
10	**Parcel Recognition**	Heading	Not Applicable
11	The robot shall be able to identify the parcel's destination when it is placed on the robot	Requirement	Accepted
12	**Parcel Delivery**	Heading	Not Applicable
13	The robot shall be able to manoeuvre the parcel into the destination bin	Requirement	Accepted
14	The robot shall be capable of carrying a parcel up to a size of 150mm^3	Requirement	Accepted
15	**Robot Design**	Heading	Not Applicable
16	The robot shall be no greater than 300mmx300mmx350mm (l*w*h)	Constraint	Accepted
17	The robot shall be capable of driving for 5 minutes using only battery power	Requirement	Accepted
18	**Battery**	Heading	Not Applicable
19	Battery management shall be implemented to regulate operation and not pose a risk of electrocution or fire in the case of a failure	Requirement	Accepted

Table 2.1 - User requirements table

3.0 Design Overview

The design of the robot was decided on following the user requirements.

The robot utilises Raspberry Pi 4 as its central controller, Pi-4 is embedded with a Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz [2]. The SoC contains Bluetooth, Wifi modules and offers dynamic functionalities with its 27 GPIO pins. The Pi is open source, hence accessible for further development. Overall, the micro-computer is cost-effective, reliable and efficient, hence very suitable for this task.

To reliably locate itself inside the arena, the robot has been equipped with multiple time of flight (ToF) sensors, which are able to accurately determine distance to objects. The robot will be able to utilise the distance the ToF sensors receive to navigate the arena.

To be able to communicate with other robots, the bluetooth feature of the on-board Pi is utilised. As the blue-tooth module is a popular power-efficient method of communication, it is widely used by other robots. Given the two robots are sharing the same communication architecture developed in section 3.7, the other robots will be able to connect to the robot designed in this report and share essential information needed to complete this task.

The delivery system of the robot will be responsible for unloading the parcel at the destination. The delivery system consists of a slanted plate with a gate, installed at the top of the robot, on which the parcel is stored. When the robot reaches the destination for a parcel, the gate is actuated by a servo, and is lowered, which allows the parcel to slide out, and drop into a collection basket.

The robot uses RFID tags for parcel recognition, a cheap and reliable method for which different objects can be identified. An RFID tag reader is connected to the Raspberry Pi, and allows RFID tags to be read. Given there are three different destinations for the parcels, RFID tags will be encoded to store one number from 1 to 3, each number corresponding to a destination. When the Raspberry Pi reads a number from the tags, it will match the number to a destination, and proceed to plan a path towards the destination.

3.1 Block Diagram

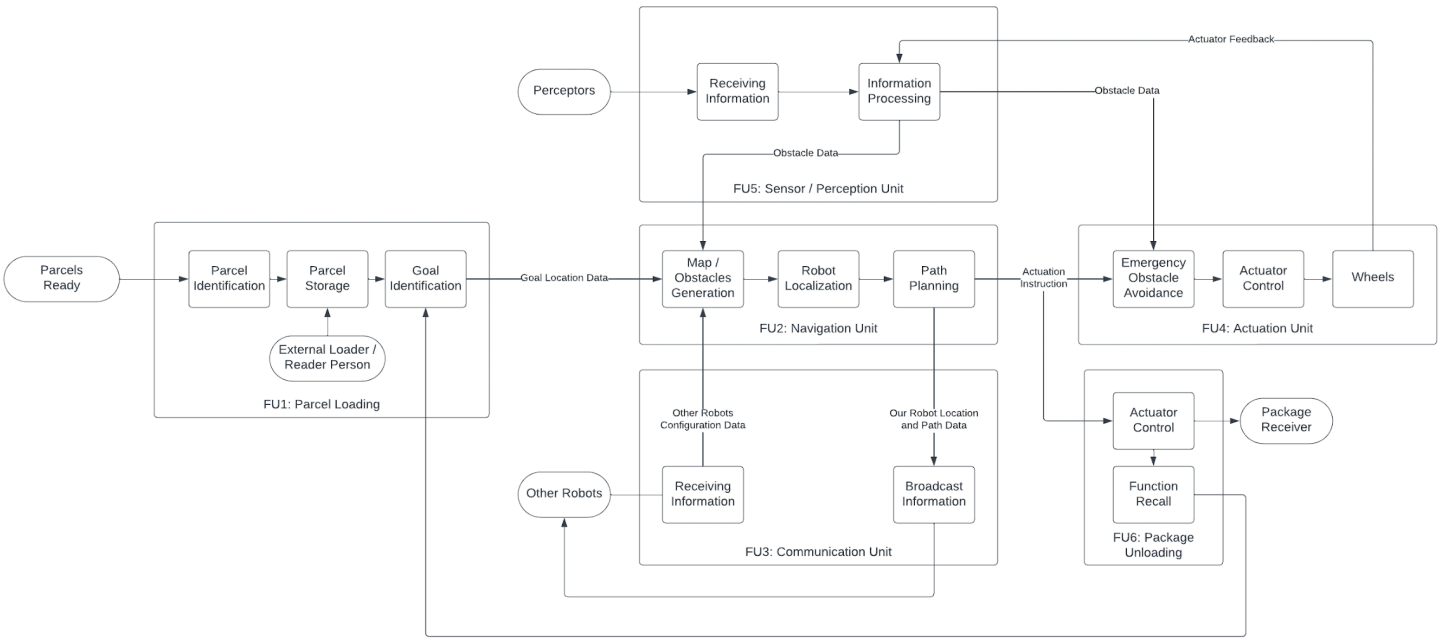


Figure 3.1 - Block diagram of system integration plan

3.1.1 - Parcel Loading

Parcels will be loaded onto the robot by a team member, multiple parcels to the same delivery destination can be loaded at the same time. A panel controlled by a servo will act as a gate to stop the parcels from sliding off the robot and will lock the gate shut as the robot moves around. This will allow the robot to carry multiple parcels without having to worry about the total weight of the payload. This will be discussed more in detail in section 3.5.

3.1.2 - Navigation unit

The navigation unit uses the A-star path planning algorithm [3] for route planning and will be expanded upon more in section 3.4.1. The map generation code has preloaded walls of the arena, therefore as long as the localisation operates successfully with minimum error, the collision between robot and the walls is unlikely. Further detail will be discussed in section 3.4.5.

3.1.3 - Communication Unit

As the cooperating team was also using a Raspberry Pi for processing, the inbuilt bluetooth functionality of Pi's allows bi-directional communication between two Pis, using the Python package "Socket" [4] for communication. The data passed between two robots include: the state of the robot, current coordinates and rotation angle. Further detail will be discussed in section 3.7.

3.1.5 - Sensor/Perception Unit

A mpu9250 inertial measurement unit (IMU) was used for measuring the rotation of the robot, utilising both vl53l0x and vl53l1x time of flight (TOF) sensors for distance measurements. These measurements were also used for the localisation of the robot. The sensors will be discussed in detail in section 3.2 below.

3.1.6 - Package unloading

The package unloading system reflects the parcel loading section, reversing the direction of the servo when the correct coordinates have been reached and movement has ceased.

3.2 Control System and Sensors

3.2.1. Motor Encoders

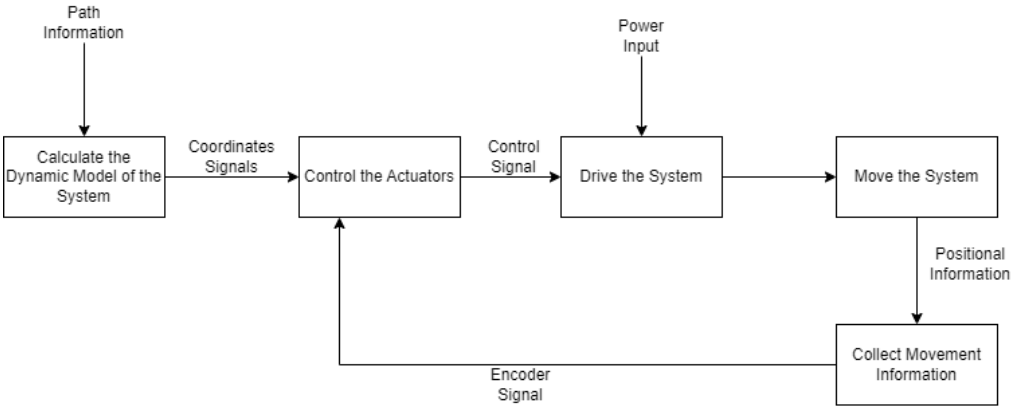


Figure 3.2.1.1 - Motor encoder block diagram

The motor encoders are used to measure the motors velocity, which is utilised in controlling the motors' velocity in alignment with the target velocity specified by the central dynamical model (Computed by Raspberry Pi). This control system utilises an Arduino Nano as its core motor controller, which has several advantages:

1. Low power consumption of 95mW.
2. Affordability.
3. The capability of assigning hardware interrupts to pins as opposed to digital interrupts, which are generally faster and more responsive.

These attributes enable the Arduino Nano to accurately process the signal read from the encoders with minimum delay.

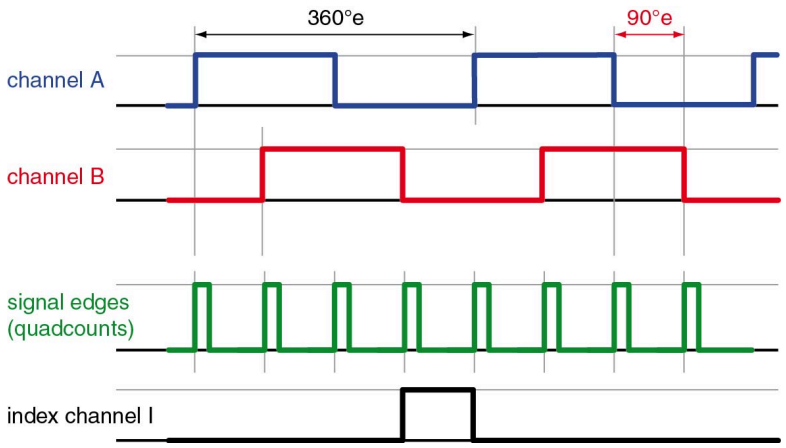


Figure 3.2.1.2 - Encoder wave diagram [5]

The angular velocity (w) of the motor is measured through estimating the time difference (Δt) between the two rising square edge of “Channel A” and “Channel B” shown on the above figure and substitute the time measurement into the following equation:

$$w = (1/\Delta t) * (2\pi/num_{rev})$$

Equation 3.2.1.1 - Angular velocity of the motor

Where num_{rev} is the number of counts of encoders measured for a full revolution of the motor. In the case of this project, it is measured to be 894.

The calculated angular velocity is then fed into a Proportional Integral (PI) controller implemented on the motor controller (Arduino Nano). The PI controller utilises a proportional constant of 10 and an integral constant of 8.

PI controller is a type of commonly used control strategies, compared to PID controller. It is more robust against noisy signals, as it does not have a derivative component. The proportional component of the controller improves the response time and the integral component of the controller will improve the rise time in the Linear Time Domain, as well as eliminating the steady state error of the controlled signal.

However, PI control, like all other controllers with integral components, suffers from the problem of Integral windup. This means the error will continuously accumulate if the controller is tasked with reaching states it cannot reach. Ultimately, resulting in endless oscillation at zero velocity. This problem is addressed through implementing a zero-state attenuation: when the PI controller is signalled to reach a velocity of zero, the PI controller will become a P controller, therefore nullifying all of the previously accumulated errors. Doing so will also not impact on the accuracy of the system, as the difference in the expected and measured velocity is already compensated for by the dynamical model in the master planner.

3.2.2. Time of Flight Sensors and Inertial Measurement Unit (IMU)

Component Name	I2C Bus	I2C Address	Measure Mode	Pins
vl53l0x	I2C0 (SDA0, SCL1)	0x21, 0x22, 0x23, 0x24, 0x25, 0x26	High Accuracy	GPIO(14), GPIO(19),GPIO(21), GPIO(28),GPIO(27), GPIO(26)
vl53l1x	I2C0 (SDA0, SCL1)	0x31, 0x32	Short Distance	GPIO(2), GPIO(3)
mpu9250	I2C1 (SDA6, SCL7)	0x68 (Gyroscope + Accelerometer), 0x18 (Magnetometer)	AFS_SEL=0 FS_SEL=2 [6]	NA

Table 3.2.2.1 - Sensor data

The Time of Flight (ToF) Sensors and the Inertial Measurement Unit (IMU) are managed through a Raspberry Pi Pico micro-controller. The Pico micro-controller utilises the community managed EP Arduino-Pico core [7] to compile and run the Arduino Code. Compared to running CircuitPython directly on Pico, the Arduino-Pico core is much faster[8]. Additionally, being an Arduino Core also means the system is able to have access to the abundant Arduino Hardware Libraries.

RP2040 on Pico being a dual core chip also supports two I2C busses and multiprocessing. This allows for the distribution of workload and data integrity of I2C communication.

3.2.2.1 IMU data Processing

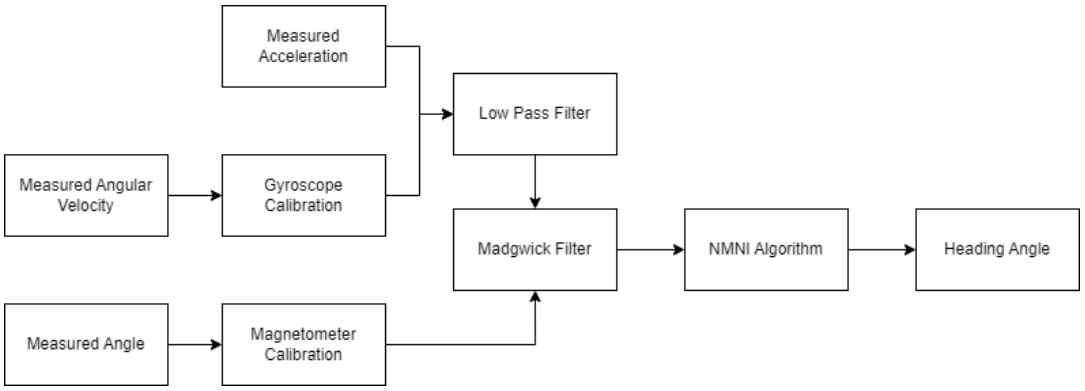


Figure 3.2.2.1 - IMU data processing block diagram

To implement sensor fusion and convert the raw measured data into AHRS estimation of the robot, a Madgwick filter is used. The Madgwick filter was originally proposed in 2010 by Sebestian Madgwick[9]. The filter employs quaternion representation of orientation and

objective function gradient descent in the optimisation process. Madgwick Filters are believed to be faster than traditional Kalman Filters and more accurate in estimating the yaw angle compared to Mahony Orientation Filters [10].

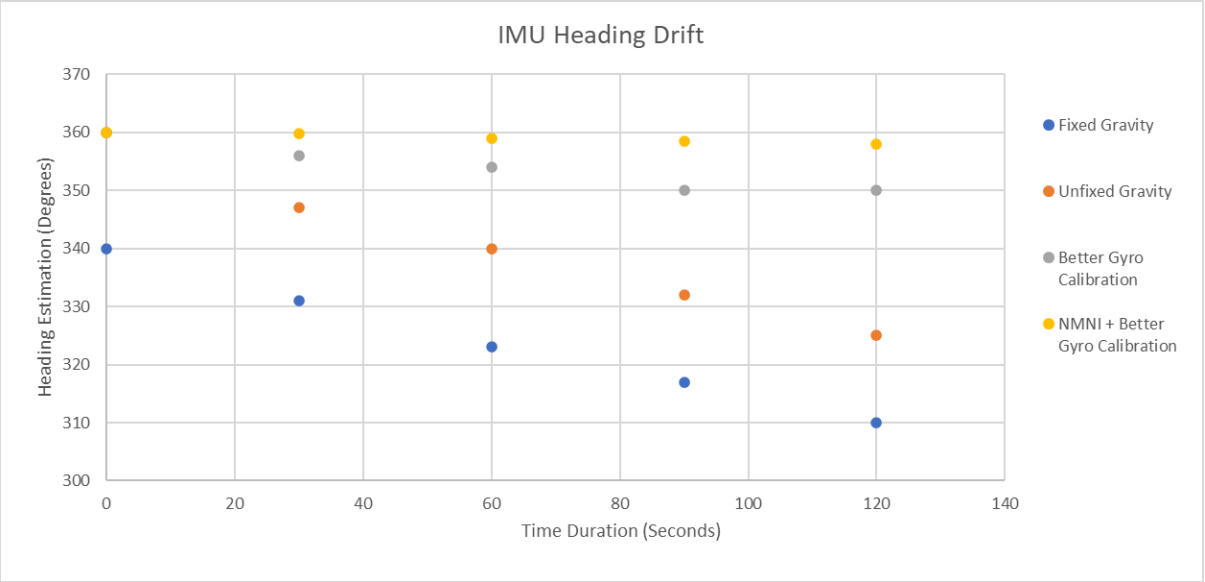


Figure 3.2.2.2 - IMU heading drift vs time relationship

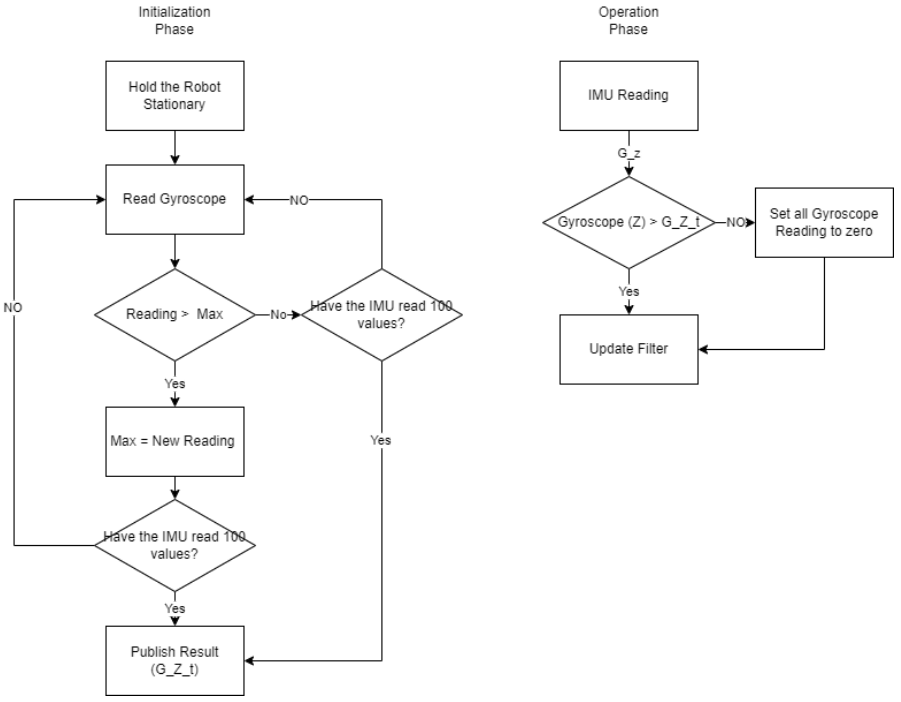


Figure 3.2.2.3 - Block diagram for IMU readings

However due to the presence of random noises inside the Gyroscope readings, the continuous integration of the Madewick filter can lead to the accumulation of error, hence

yaw angle will drift over time. This is evident in the figure 3.2.2.2 shown above, without any compensation methods (Unfixed Gravity) the drift in yaw angle can be as much as 35 degrees in 2 minutes. To resolve this problem, different approaches have been tested and it is found that implementing better Gyro Calibration through Increasing the number of Gyroscope reading during the calibration to 128 times as oppose to 64 times and implementing the design team's interpretation of NMNI algorithm (shown in the flow chart above) proposed in Hoang's paper [11] yields the best result. Ultimately with proper on-site calibration, the IMU is able to yield a minimum drift of approximately 2 degrees for the 5 minutes of operation.

3.2.2.1 ToF

Time of Flight Sensors (VL53l0X and VL53l1X) have the following advantages and disadvantages compared to the ultrasonic sensors (HC-SR04) used in Milestone 1:

Advantages	Disadvantages
- Higher accuracy, longer range and higher resolution. - Not affected by the temperature and pressure of the operation environment. - Occupies less space. - More robust against collisions and impacts.	- Light may be washed out in an environment that is extremely bright. - A bit more expensive compared to traditional ultrasonic sensors.

Table 3.2.2.2 - Pros and cons of ToF sensors

The original thinking behind using ToF sensors is due to the original design requirement of robots needing to cooperate with randomly allocated teams. Since a large number of groups will be using ultrasonic sensors, the ToF sensor will not be affected or interfered by the other robots' sensors during obstacle avoidance.

However the requirement of the project changed in week 9 to allow for pre-agreed collaboration, this allowed the obstacle avoidance aspect of the project to be manageable by the communication. ToF sensors tasks as of now are mostly for localisation which is discussed in detail in Section 3.4.5.

3.3 Power Systems

3.3.1. Battery+ Subsystem

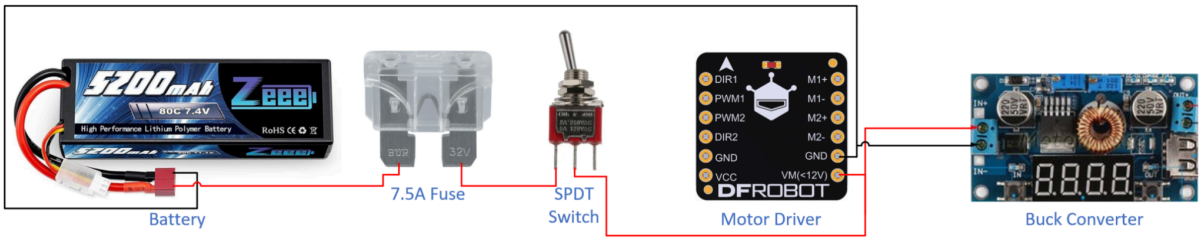


Figure 3.3.1 - Battery system circuit diagram

The robot has three separate voltage circuits powering separate sub-circuits. Power to the robot is supplied by a 2 cell 5200mah LiPo battery. The power source has an inline switch to allow us to shut down the robot when it is not active and also an inline fuse to protect the circuitry and users in the event of a short circuit. The battery directly supplied power to the motor driver and the buck converter is used to power the 5V subsystem.

5V Subsystem

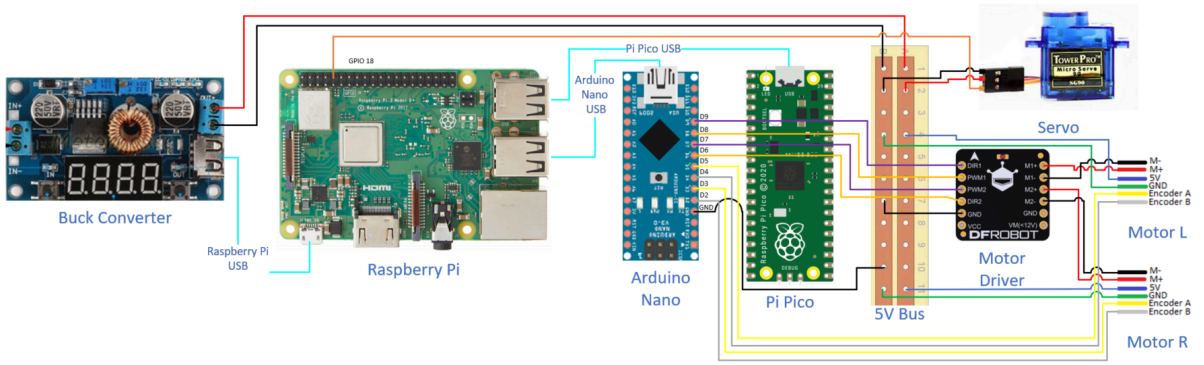


Figure 3.3.2 - 5V System Diagram

The 5V subsystem is powered through a buck converter and supplies power to the raspberry pi, and acts as a power source for the motor encoders and servo. Power is also supplied to the Raspberry Pi Pico and Arduino nano through the Raspberry Pi's USB ports. This has been done to reduce wiring as we can combine the robot's power and serial communication between the devices into one wire. A 5V bus is employed to provide power to the other components not connected by USB. This bus also provides the robot with a ground reference, allowing us to have a common ground between all power sources thus eliminating the chance of a floating ground. This also includes the motor driver wiring which interfaces with the microcontrollers.

3.3V Subsystem

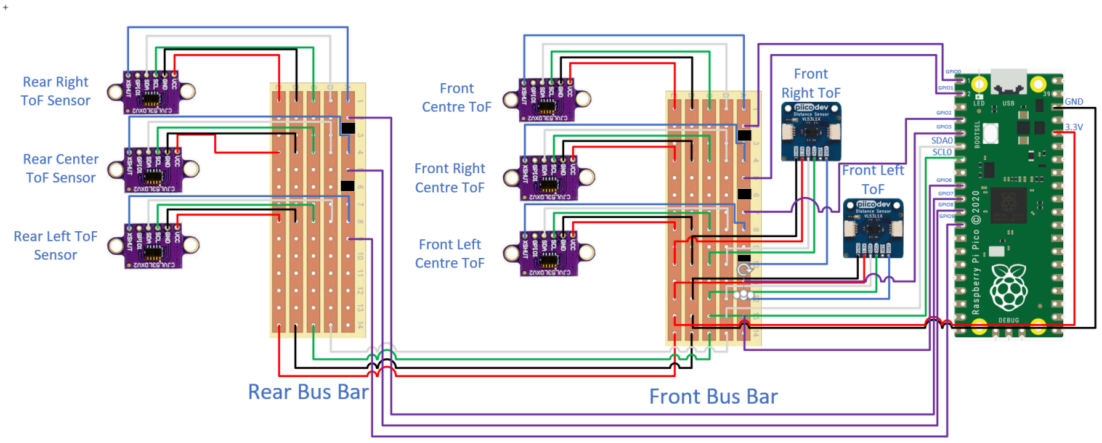


Figure 3.3.3 - 3.3V Subsystems Circuit Diagram

The 3.3V subsystem is driven directly from the Raspberry Pi Pico. As this is used to power the time of flight sensors, the 3.3V output of the Raspberry Pi Pico is sufficient to power all of the sensors. This power and subsequent I2C bus used to communicate with the sensors is split across two buses one for the front time of flight sensors and one for the rear. The diagram above also shows the shut wires that are attached to the time of flight sensors that are used for the initial address assignment.

3.4 Navigation and Localisation

The original design for the robot's navigation system involves generating a map filled with waypoints and obstacle data. This allows the system to use path planning algorithms such as A-Star (A*) to trace a theoretically best path towards the end goal. As such, assuming that an accurate representation of a real-time map can be constructed, four main subsystems are required for navigation to successfully function:

1. Waypoint Generator (A* Algorithm)
2. Point-to-Point driving algorithm (Tentacle Planner / Model Predictive Control)
3. Obstacle Avoidance System
4. Robot Localization (Homography / Wall-based Localization)

The code for the waypoint generator, tentacle planner, homography and obstacle avoidance can be found on the github for the project, found in Appendix X.

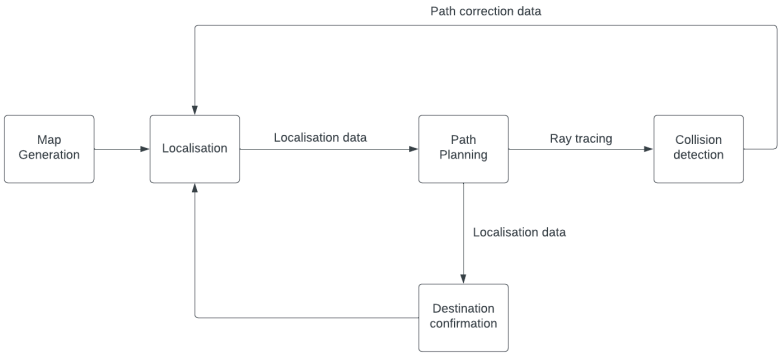


Figure 3.4.1 - Navigation and Localization System block diagram

3.4.1 - Waypoint Generation using RRT and A*

In milestone one, the robot's main navigating system revolves around using rapidly-exploring random trees (RRT). While it is simple to implement and fast to operate, the path it comes out with can be inconsistent and random, and often not the most efficient. As a result, the planner often adds many unnecessary hooks and turns to the path and can be an annoyance to test with.

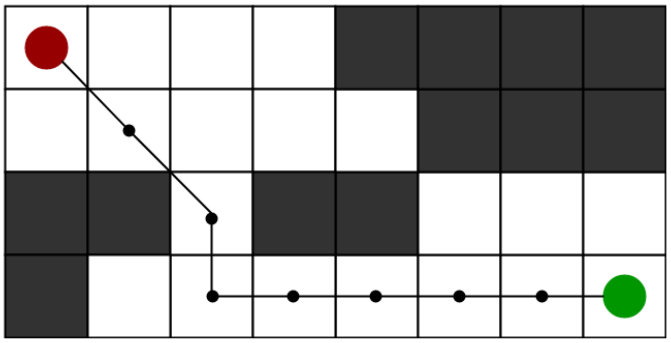


Figure 3.4.2 - Grid-based A Algorithms, path found by the robot are guaranteed to be shortest and avoid obstacles. [12]*

Thus, for the final design, the team decided to implement the A* Algorithm based on the implementation done by PythonRobotics[13]. The A* Algorithm is a graph-traversal and path search algorithm. It is slow and bulky, being an $O(b^d)$ space complexity, but guarantees the shortest path to be found.

The main limitation of the implementation of A* is how it currently sticks to a grid. As such, the waypoints generated can look rather rigid and non-smooth in real life. PythonRobotics has an implementation of Hybrid A* which allows the algorithm to process between grids, but for the purpose of the project this was not explored.

3.4.2 - Point-to-point Navigation using Tentacle Planner

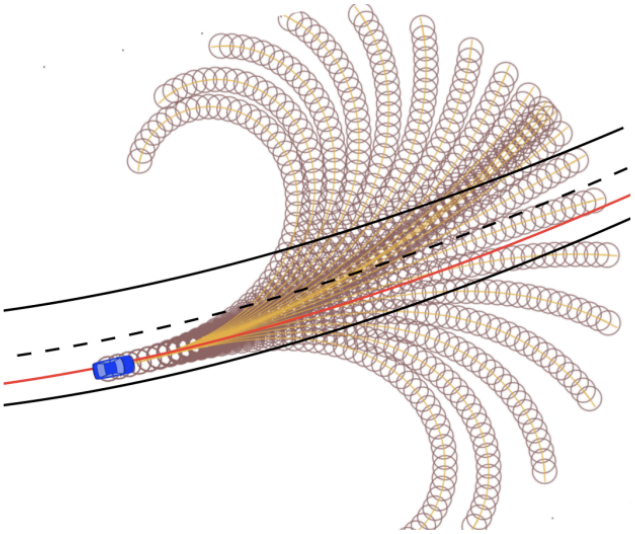


Figure 3.4.3 - Model Predictive Control on Vehicles. [14]

The tentacle planner, a simplified form of model predictive control, is a cost-based algorithm which controls the linear and rotational velocities of the robot. It generates a series of trajectories stemming from the robot's position, where each trajectory is a prediction of the robot's future position using an associated linear and rotational velocity.

Using this prediction, the algorithm would rank each trajectory using a cost function. This cost function would select the trajectory that is closest to the desired goal point, which is the next waypoint. It also discards any trajectories that would directly collide with an obstacle.

A big limitation of the current implementation is that it only looks forward by a single time step. More advanced versions of model predictive control would look forward to multiple steps to obtain better resolution and forecast. As a result, the current implementation can be described as a relatively "greedy" approach, which can sometimes result in the algorithm being stuck in a local minimum cost function region (in cases where it needs to increase the cost function to lower it further), therefore causing the robot to fail to reach the next waypoint. This was mitigated by increasing the amount of waypoints created through the A* planner, and greatly reduced the speed of the robot so it can be more precise with its speed.

3.4.3 - Obstacle Avoidance

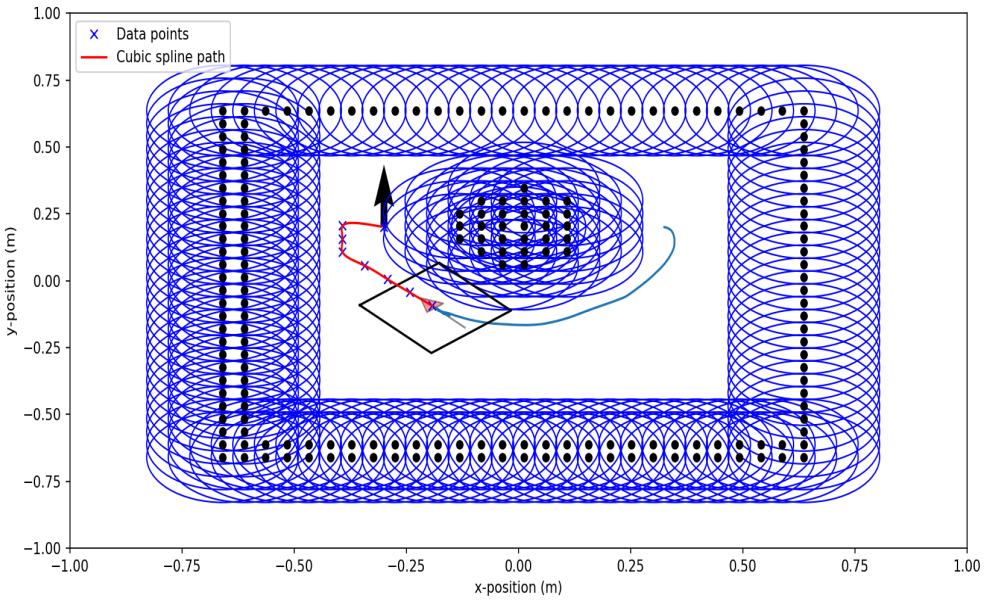


Figure 3.4.4 - Obstacles on map with robot driving. Obstacles have a radius that the robot needs to keep away from at all times.

The majority of obstacle avoidance occurs within both navigation algorithms with A* and Tentacle Planner. For both navigation systems to function, an accurate obstacle map is required. In milestone 1, the robot utilises a dynamic obstacle detection system where the ultrasonic sensor would create obstacles as it traverses the map, thus filling the arena with more and more obstacle data along the way. However, in practice the robot's position was not accurate enough to be trusted, and this resulted in obstacles being generated everywhere, rendering the system unusable.

Since the only other obstacle in the arena is cooperating robots, real time robots are utilised to pose data as obstacles through the robot's communication systems. However due to time constraints, this was not implemented. Instead, a state-based design was implemented to ensure both robots would not collide ever by being as far away from each other (See Section 3.7).

In addition to this, due to needing to get very close to the wall to deliver packages, the obstacle avoidance precautions often get in the way as the robot finds itself unable to path near the obstacle. In those situations, the obstacle avoidance system is outright disabled.

Therefore, in the final design, it may be considered that there wasn't even any obstacle detection systems made to the design, but rather carefully tuning goal points, coordination with the other team and utilising map localization to ensure the robot would not collide with anything. As such, the lack of an obstacle avoidance system can be nullified.

3.4.4 - Camera pose estimation using homography

A camera pose estimation homography is used for homography for this project. That is, a camera will be installed facing directly upwards, and a homography algorithm will be run to estimate the current position of the robot in terms of a reference image and will include translational and rotational information.

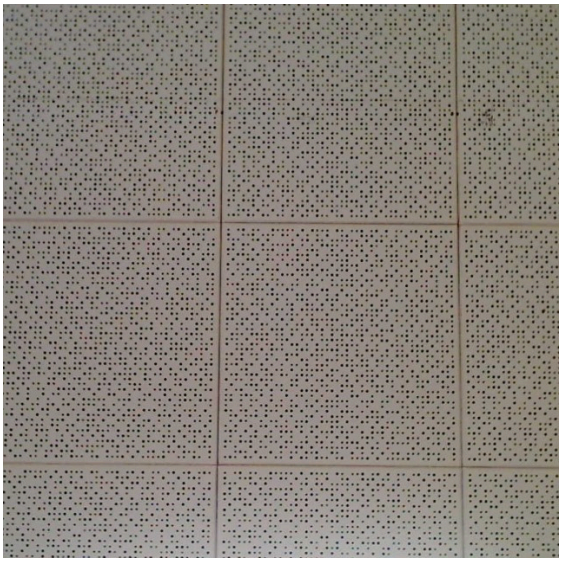


Figure 3.4.5 - Example reference image from the camera

Using just wheel encoders and TOF sensors, there will inevitably be some drift in the accuracy of the localization, and that is what this method of localization will aim to account for. The algorithm will compare both images and draw matches between like features in both pictures and attempt to estimate the movement in the camera.

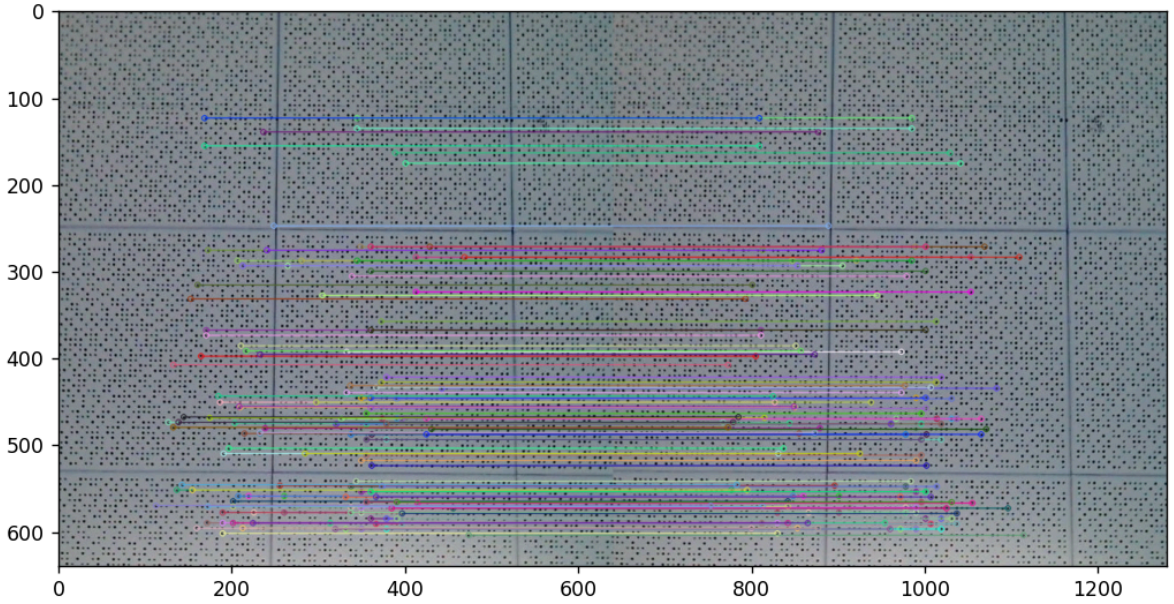


Figure 3.4.6 - Example of matches between to images

However, for homography to be accurate, the images will either need to have very distinct features, or there is not much movement between the reference and the test image. The ceiling of the competition environment has a very uniform and repetitive pattern, so the program has a lot of trouble detecting the true location of the camera, hence yield a very low accuracy for its estimates.

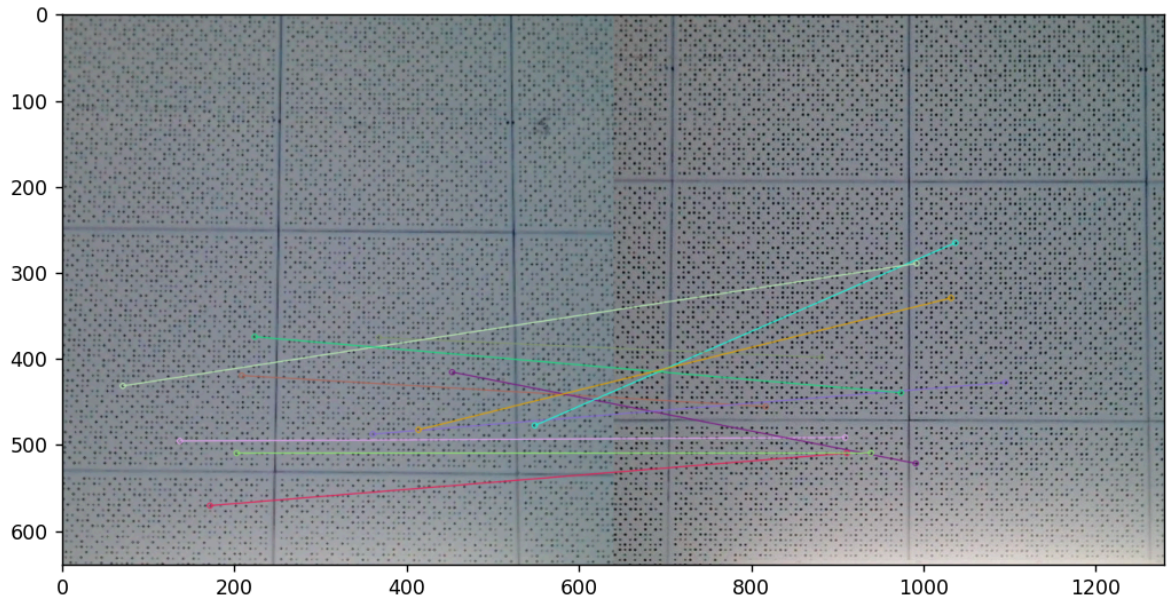


Figure 3.4.9 - Matches against a 200mm horizontal translation and 50mm vertical translation

```
The angle of rotation is -90.99 degrees.
The translation is x=396.32, y=261.30 pixels
```

Figure 3.4.10 - Incorrect translation information provided from the 200mm horizontal and 50mm vertical translation homography

To combat this, some preprocessing of the images was required before the images could be sent through the algorithm.

As there were many holes in the ceiling tiling which were in the exact same position on each tile, it was decided that only the edges of each tile would be used for the homography. To accomplish this, the images were run through a series of filters.

A blur filter is the first filter that the images will be put through, hopefully filtering out some of the holes in the tiles with an aggressive enough filter. However, a filter too aggressive would also filter out the edges of the tiles too, so some residual holes were left over from the blur filter.

The types of blur filters that would be implemented are numerous, but the most common types are listed below:

1. **Averaging**: Given a kernel size, the algorithm will convolve the image with a box filter, essentially getting the average of a group of pixels and setting the entire group to be that colour and will output a blurred average of the original image. A larger

kernel size means that each group of pixels which are average is larger than with a smaller kernel. A larger kernel would mean more aggressive blurring and less detail being retained.

2. **Gaussian**: Similar to an averaging filter, but uses a Gaussian kernel instead of a box kernel, and weights the middle of each kernel heaving than the outside and therefore should result in a more accurate representation of the initial image. However, for the purposes of this project, a Gaussian blur is unwanted as it will not filter out the holes in the tiles as well as a box filter
3. **Median**: Similar to averaging, but instead of using average, uses median instead and replaces the central element with this median colour. Effective at removing noise. As the kernel size for a median filter needs to be an odd integer, this filter is not as flexible as other filters.
4. **Bilateral**: A step further than Gaussian filter, firstly doing a Gaussian filter for the surrounding pixels, then taking another Gaussian filter which is a function of pixel difference, ensuring that only pixels with similar intensities to the central pixels are used for blurring. This retains sharpness of lines and is very good at denoising, however will take longer to run as two Gaussian filters will need to be used.

An averaging filter was used as it provided a good balance of computational cost, accuracy and best fit with the filters to follow.

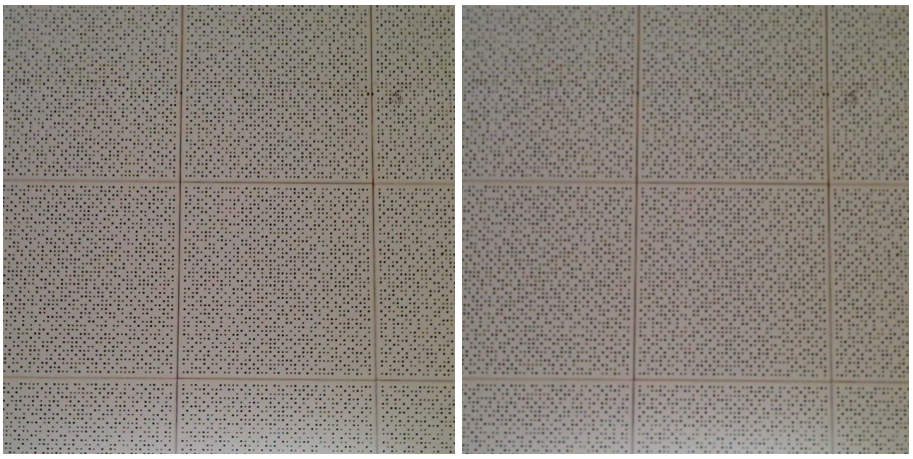


Figure 3.4.11 - Original image(left) and blurred image (right)

Although it looks like not much has been changed, the subtle blurring of the lines will be pivotal in the other filters that will be implemented after the blur.

The blur filter was implemented with a kernel size of 2, a value obtained through trial and error to produce the best results. A larger kernel size would result in a more aggressive blurring of the image and could likely result in the edges of the tiles being filtered out and all the important information being lost between filters, the Canny filter that is to be implemented after the average filter not being able to detect the lines if the blurring is too aggressive.

Next, the processed image was run through a denoising filter to further remove the holes. The python function fastNlMeansDenoising was used to denoise the image, utilising a nonlocal means algorithm for the denoising. It returns a monochrome image which is then sent through to the next filter.

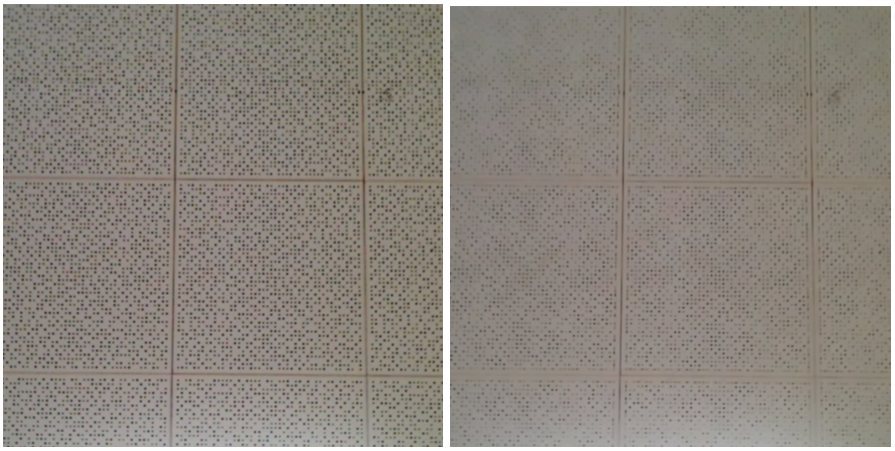


Figure 3.4.12 - Blurred image (left) and denoised blurred image (right)

As seen above, many of the holes are removed from the image, but leaves the tile edges grainy and out of focus, this is due to the aggressive denoising filter applied to the blurred image. However, there is enough detail in this image to send through to the next filter.

Finally, a Canny filter is applied to this final image for edge detection purposes. At this point, the image will have all the holes blurred enough such that the Canny filter only picks up the edges of the tiles. The Canny filter will have a threshold of detection where edges will be picked up. The higher the threshold, the less edges will be detected and therefore more noise will be eliminated. Through trial and error, a filter threshold of (0,150) was chosen as the best performing.

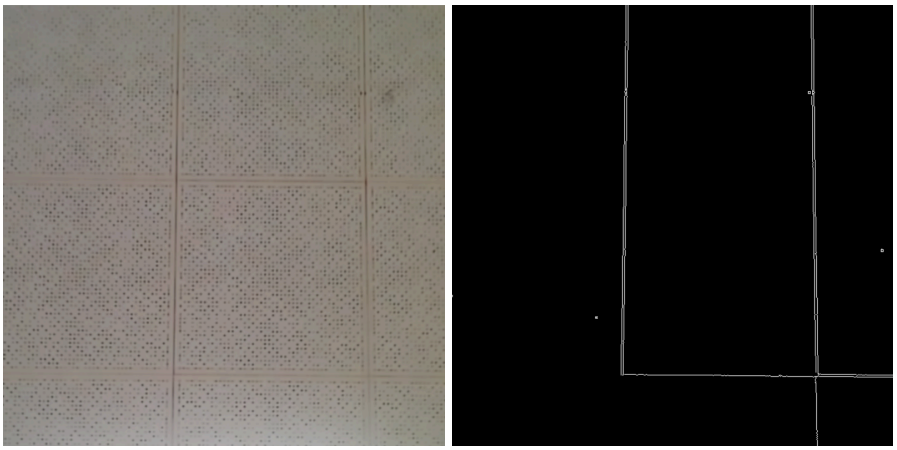


Figure 3.4.13 - Denoised image (left) and Canny filtered image (right)

This provides a good resulting image with sharp lines and removed holes.

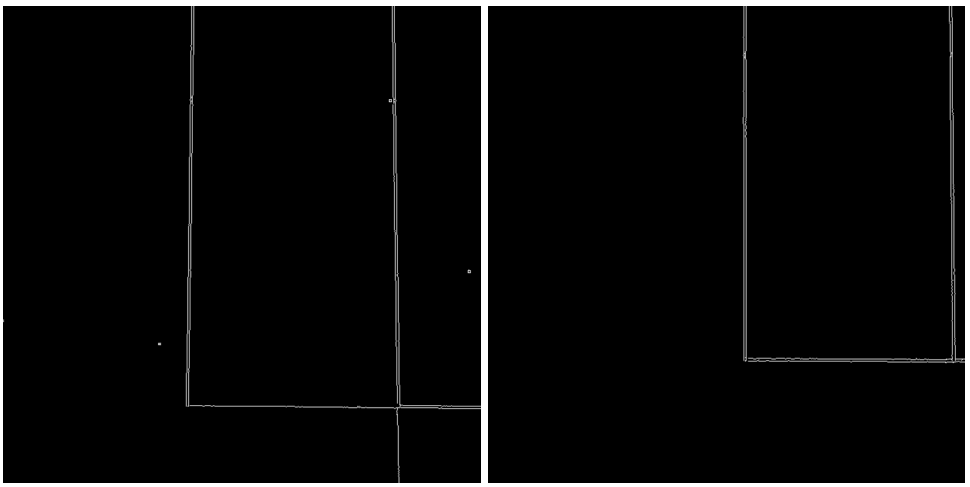


Figure 3.4.14 - Original image with filters (left) and original image with translation with filters (right)

However, this image still had a large amount of error while estimating.

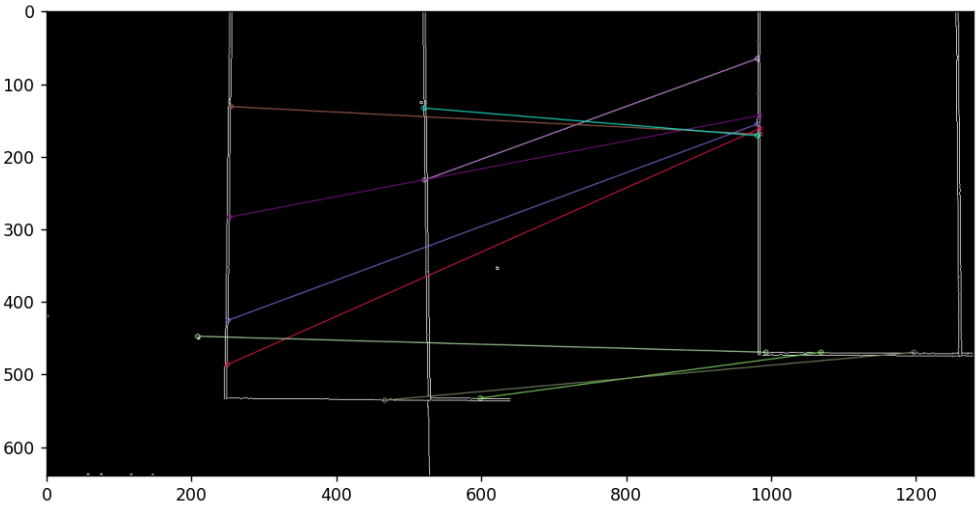


Figure 3.4.15 - Matches between filtered images with translations

```
The angle of rotation is -96.04 degrees.
The translation is x=329.79, y=129.92 pixels
```

Figure 3.4.16 - Incorrect camera pose estimation

However, the homography is accurate with small movements of the camera.

As the robot's navigation works, it means that there will be a prolonged amount of time spent in the middle of the arena and therefore that is where the initial reference image will be taken. In the first iteration of the navigation, the robot will take a picture in the middle of the arena and this is the reference image that the robot will use to do camera pose estimation on. This localisation is used to correct for drift, and as the IMU can provide very accurate angle measurements as discussed in section 3.2.2, only correct translational data is required to be provided from the homography.

3.4.5 - Using sensor data for localisation

Similar to using homography in the middle of the arena to account for drift, the sensor method activates its localisation when told by the navigation (Table 3.8.1), that the robot is in the middle of the arena. Utilising sensors in front and to both sides of the robot, the distance from each wall and the precise angle the robot is provided by the IMU, the algorithm is able to estimate the true location of the robot. For this method, the internal measurements of the robot had to be taken into account as the position of the robot was the most accurate when measuring from the centre of the base plate.

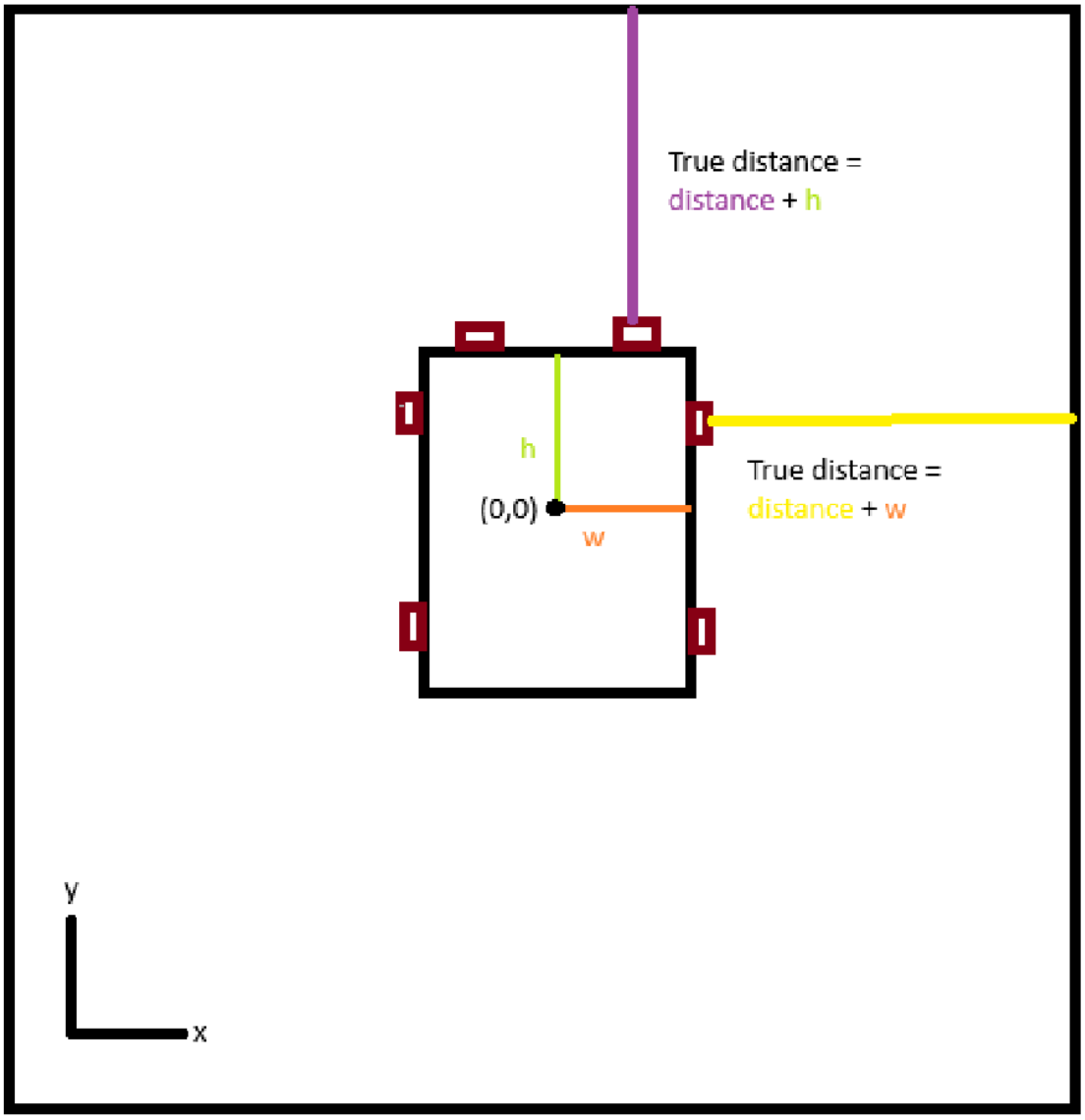


Figure 3.4.17 - Visual representation of the localisation process

Using multiple sensors for each degree of freedom, the measurements for each direction were averaged out across each sensor to hopefully balance out the biases in the sensor. The Pi would then run an algorithm to convert these measurements into a coordinate following the above coordinate system with the centre representing the origin. This

coordinate would then be sent back to the navigation system and the current position would then be updated with the one obtained from the localisation.

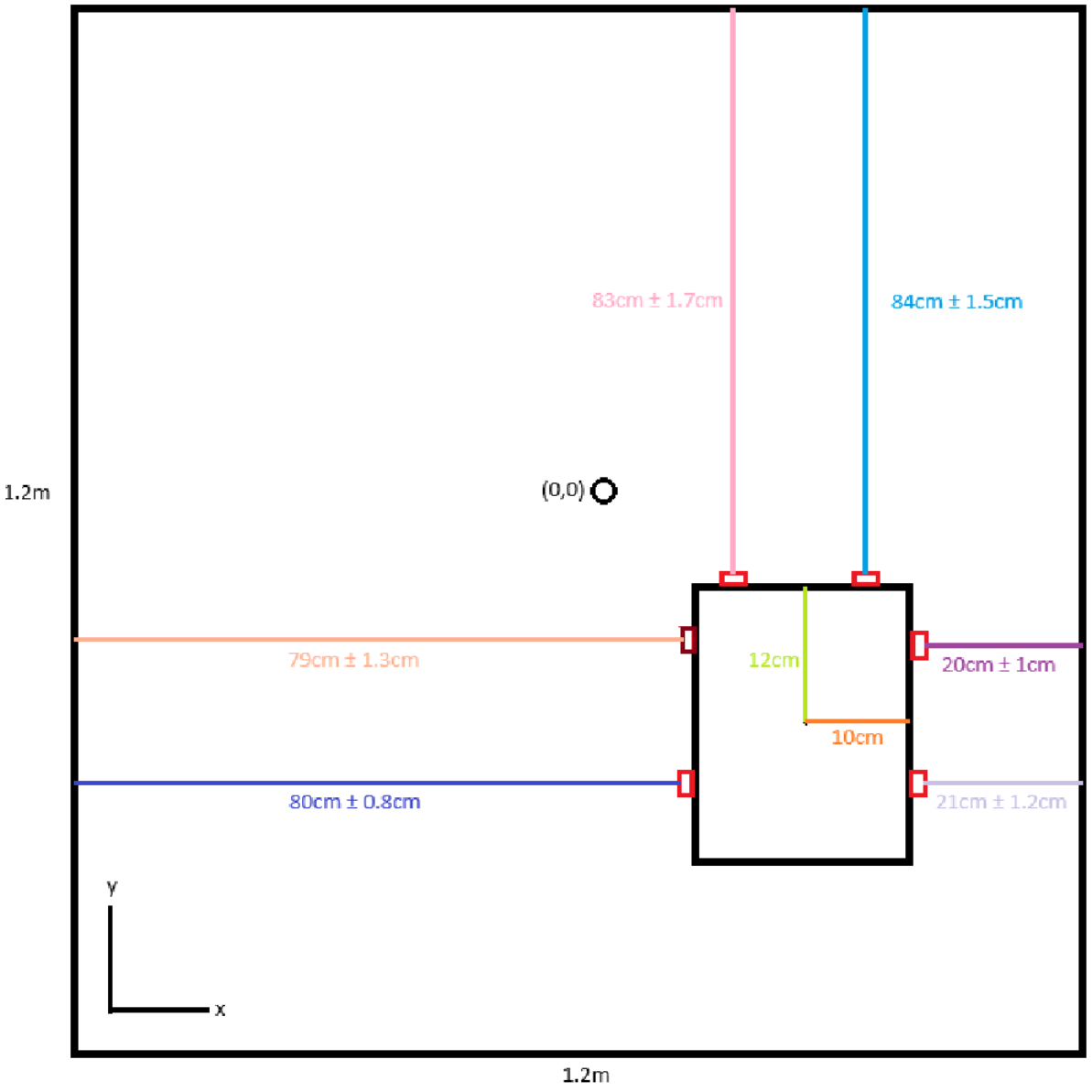


Figure 3.4.18 - Example of sensor measurements including variances

Above is an example of the sensor information being fed into the CPU. To calculate the actual coordinates of the centre of the robot, the following formulae were used:

$$y\,coordinate = 60 - (front_1 + front_2)/2 + h$$

Equation 3.4.1 - Coordinate calculation utilising two front sensors in cm

$$x\,coordinate = (60 - (left_1 + left_2)/2 + 60 - (right_1 + right_2)/2)/2$$

Equation 3.4.2 - Coordinate calculation utilising two left and two right sensors in cm

3.5 Mechanical Design

The mechanical components of the robot are responsible for delivering the parcel into the parcel delivery bins. As seen in figure 3.5.1, the mechanical design consists of a slanted platform, and a gate that can be lowered. The parcel is to be stored on the slanted platform, and held in place by the gate.

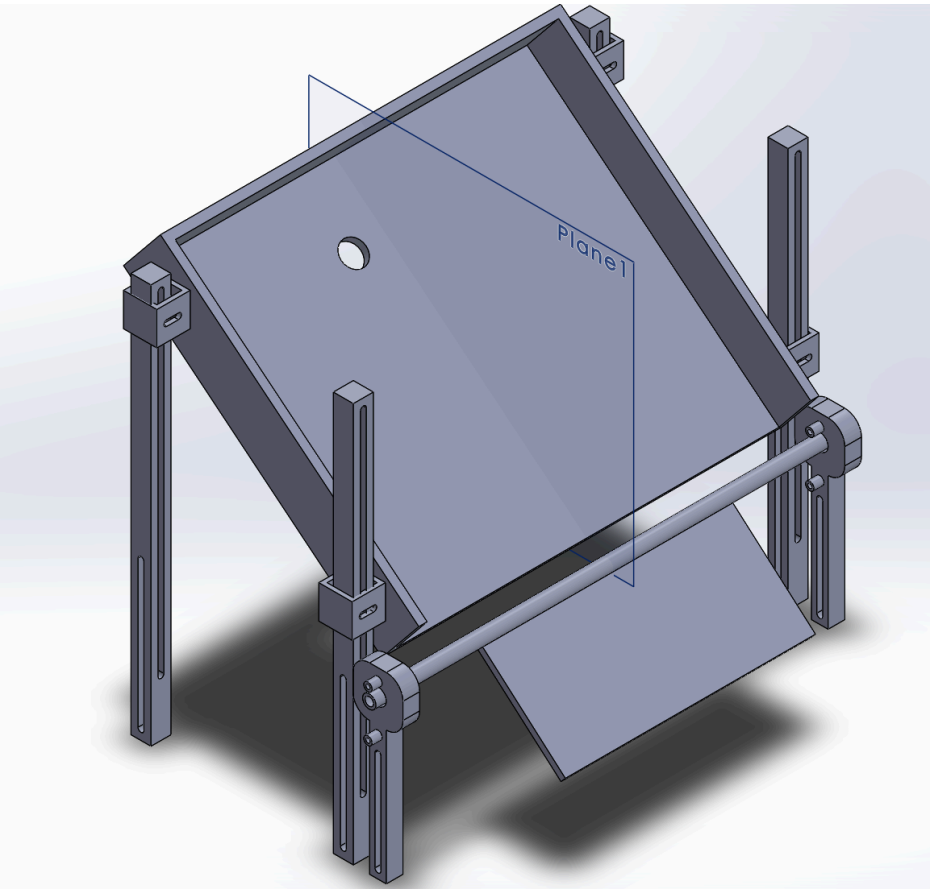


Figure 3.5.1 - CAD model of the mechanical design of the robot

Whenever the robot reaches the destination of the parcel, the robot would stop, and the gate is lowered by a servo motor controlled by the Raspberry Pi. This allows the parcel to slide off the robot and drop into the delivery bin. The process can be seen in figure 3.5.2. The servo controlled gate has been programmed to be activated whenever the robot reaches its destination. The servo rotates 90 degrees, in order for the gate to become flush with the tray, and drop the package. After 5 seconds, the servo is activated again to rotate back, and raises the gate to allow the robot to return to the loading zone.

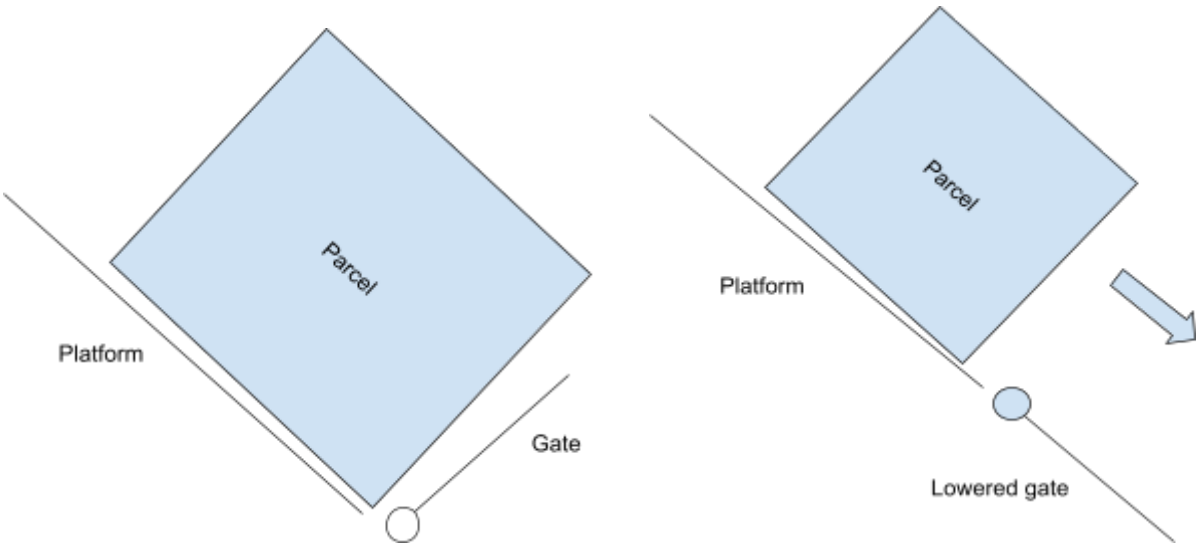


Figure 3.5.2 - Parcel held in place by the gate (left), the parcel sliding off on the lowered gate (right)

An issue that was discovered during testing for the robot is that the gate was too heavy for an unpowered servo to keep upright. When a relatively larger parcel is loaded into the robot, or whenever the robot comes to a rapid stop, the force exerted on the gate was too large for the servo to manage when unpowered, and the gate would fall open mid-travel. At best, this poses a potential collision hazard as the opened gate protrudes in front of the robot. At worst, the parcel loaded on the robot would fall out into the arena, making the delivery unsuccessful, and the parcel would become an obstacle.

In order to rectify this issue, the servo was programmed such that whenever a parcel is loaded on the robot, the servo is constantly powered and continuously drives in the direction that keeps the gate closed. This means that although the issue of the gate falling open is resolved, having the servo constantly driving the gate shut produces a great strain on the servo.

3.6 Parcel identification

The parcel identification of the robot is done through the robot reading RFID tags. An MF-RC522 RFID reader is connected to the Raspberry Pi, and installed on an easily accessible location on the robot. This reader is both able to read and write to Mifare Classic RFID tags. The circuit diagram can be seen in figure 3.6.1.

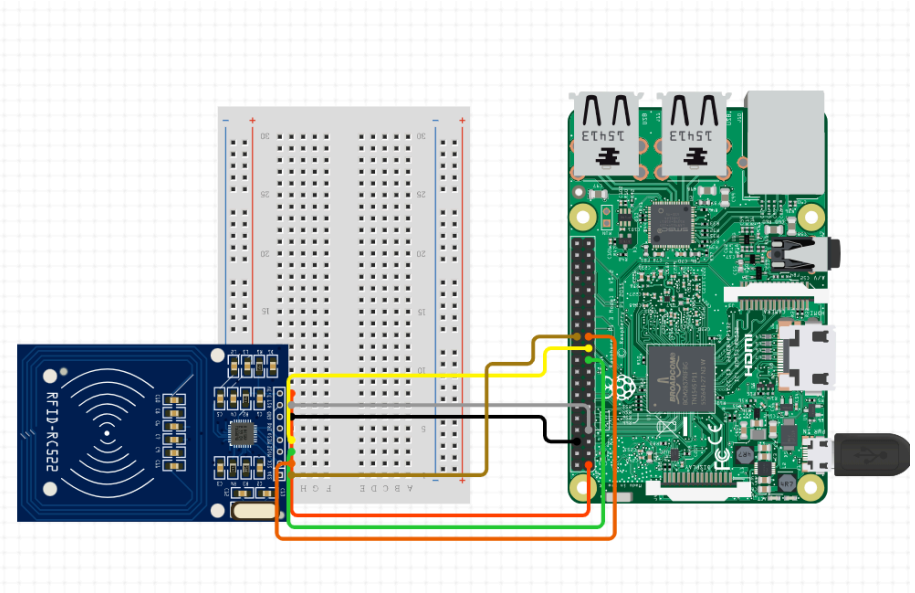


Figure 3.6.1 - circuit diagram of the RFID tag reader

The user requirements of the project dictates that all parcels will be delivered to one of three locations and therefore, all tags contain a number from 1 to 3. Each of these numbers corresponds to one of the three locations, that is, the number 1 corresponds to the first, or leftmost parcel delivery bin, number 2 corresponds to the second, or middle delivery bin, and 3 corresponds to the third bin.

These numbers can be written to the RFID tags through the MF-RC522 RFID reader module through Python. The process of writing data to the tags involves scanning the tags one by one on the reader module, which is a very inefficient process. It was originally proposed that each package would have its own RFID tags attached to it, however, given how costly and inefficient such a process would be, it was determined that having three tags with 1, 2, and 3 written to them will be sufficient.

Whenever a parcel is loaded onto the robot, the human operator would scan the tag corresponding to the robot's destination on the robot. The robot itself is pre-loaded with the physical coordinates of each of the three destinations. Thus, the robot would recognise the number scanned, match the number to a destination, and retrieve the coordinates of the destination. This would allow the robot's navigation system to plot a path towards the destination.

3.7 Inter-robot communications (Robot 33)

A combination of python's socket package and bluetooth connections allows robots to create a server and client communication protocol. This link can then be used to send messages between the two robots such as a json object. Data such as goal, robot pose and robot path were able to be sent, but in practice implementing this was complicated and not very useful to the other robot (since Team 33 was utilising a wall-following approach), thus both teams settled with a single variable state machine design.

3.7.1 State Machine of Robots

To coordinate both robots such that they are guaranteed to not collide with each other, a state machine was used to ensure only one robot is in the delivery zone and loading zone at a time.

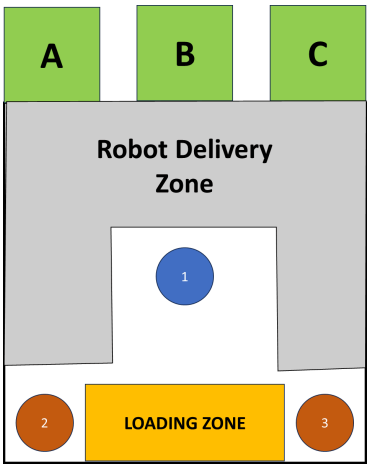


Figure 3.7.1 - Visualisation of "key zones" in the arena.

The above figure shows the designated zones on the arena our robot and Team 33's robot. Zone 1 represents the idle waiting area for our robot. Zone 2 & 3 represent the waiting area and returning area for Team 33's wall-following robot.

Since the wait zones are out of the way of each other's trajectories, it is guaranteed to avoid collisions as long as both robots only operate in the delivery zones one at a time. This also allows for one robot to load packages while the other robot delivers.

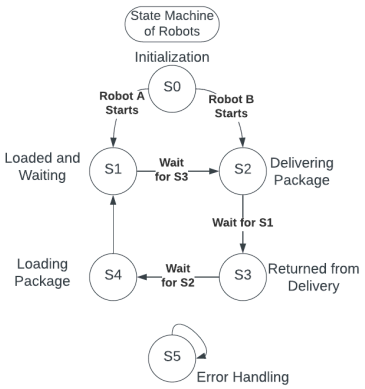


Figure 3.7.2 - State Machine of Robots.

From Figure 3.7.2, S0 represents the initialization state. S1, S3 and S4 represent waiting / loading states. S2 represents delivery state, and S5 is an error handle. Each state has a potential “wait” state where the robot idles in its current state until the other robot is complete with their actions. The following details the functionality of each state:

State 0: Initialization State

- In State 0, both robots are performing calibration, connecting to each other in bluetooth, and driving to the idle positions to prepare loading.

State 1: Waiting State

- State 1 is the default “idle” state when the robot arrives at its start position. It only enters State 2 if the other robot has returned from delivery (State 3).

State 2: Delivery State

- In State 2, the robot is delivering its package to the delivery boxes. Once it finishes delivering, it waits for the other robot to position itself at its start point before returning to avoid potential collision at the loading zone.

State 3: Returned State

- In State 3, the robot signals to the other robot that it has returned to the wait state. This means the other robot would enter State 2 into the delivery zone, at which point it is safe to go to the loading zone to load the package.

State 4: Loading State

- In State 4, the robot enters the loading zone to load a package. Once the package is loaded (signalled by an RFID input for a desired goal), it drives back to the wait zone and transitions back to State 1, getting ready to deliver once again.

State 5: Error State

- This state acts as an error handling and test state. In the event of a robot breakdown, this allows the other robot to continue operating. It is less than ideal however, due to the lack of a collision avoidance system.

3.8 System Integration

“ This robot is an Integration Hell.”

Joel Ong

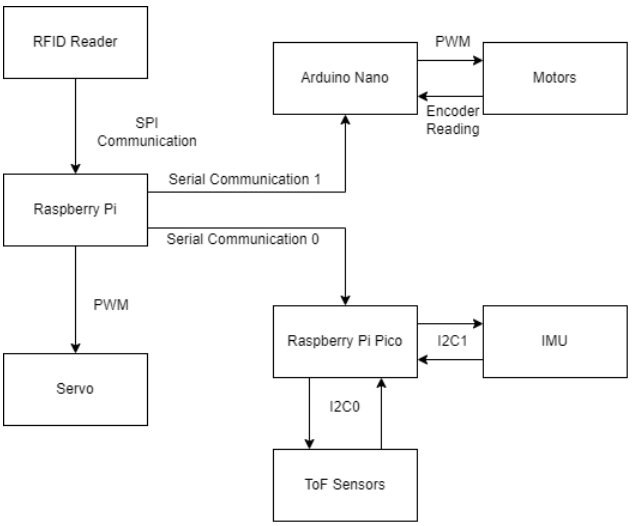


Figure 3.8.1 - Block diagram of system integration

The robot utilises three major protocols for subsystem communication: SPI, Ordinary Serial and I2C.

3.8.1 RFID Reader

The RFID reader is connected to the Raspberry Pi through GPIO pins. A Github library was imported onto the Raspberry Pi to allow for it to communicate with the RFID reader [15].

Whenever the robot is prepared to receive a parcel, the RFID reader is activated, and is able to detect when a RFID tag is put up against it. The Raspberry Pi will be able to receive the information stored on the RFID tag, which in the case of the project, is a single integer from 1 to 3.

3.8.2 I2C Communication

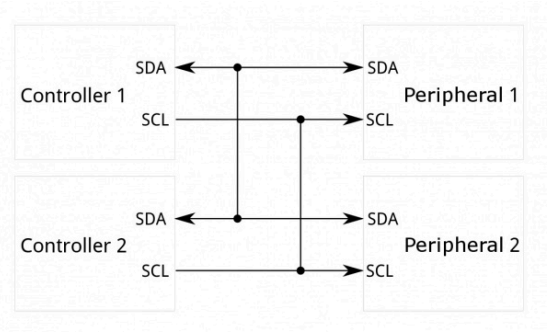


Figure 3.8.2 - I2C Bus Communication Diagram [16]

Inter-integrated-circuit or I2C communication allows for multiple devices (128 devices theoretically) to be connected to two wires (SDA and SCL). Using I2C protocol, the master (controller) devices can read and send information through the shared bus to each of the slave (Peripheral) devices.

For this project, I2C protocols are implemented as the interface between sensors (ToFs and IMU) and the Raspberry Pi Pico. Where Raspberry Pi Pico is the controller and the sensors are the peripherals. Each of the sensors are assigned a corresponding address as shown in section 3.2.2. During the integration, two I2C buses are also used to ensure data integrity during multiprocessing (so two processors will not read from the same bus at the same time). As shown in section 3.2.2, ToF sensors communicate with Pico through I2C0 and the IMU sensor communicates with Pico through I2C1.

3.8.3 Ordinary Serial Communication

The robot communicates with the two microcontrollers (Arduino Nano and Raspberry Pi Pico) through ordinary serial communication. Ordinary Serial Communication allows for transfer of large amounts of data between devices over a very short period of time(a Baud rate of 9600 corresponds to 9600 bits of data transfer per second).

3.8.3.1 Communication with Arduino Nano

A constant loop of Serial communication is established between the motor controller and the Raspberry Pi (Navigation loop). That is the Raspberry Pi will constantly publish encoded string with the following format to the Nano:

“Pi: [wl_current, wr_current, wl_goal, wr_goal]”

The Nano will decode, process this string and print a corresponding line to Serial in a similar format:

“Nano: [wl_current, wr_current, wl_goal, wr_goal]”

Where:

1. wl_current: current measured angular velocity for the left wheel.
2. wr_current: current measured angular velocity for the right wheel.
3. wl_goal: target angular velocity for the left wheel.
4. wr_goal: target angular velocity for the right wheel.

3.8.3.2 Communication with Raspberry Pi Pico

As discussed in section 3.2.2.1, the sensors are no longer required to be activated consistently. Hence, the Communication between the Raspberry Pi and the Pico are conducted through a “request and response protocol” as opposed to the constant loop. Whenever the Raspberry Pi requires certain information from the Pico, it will send a request to the pico in the form of encoded digits as shown in the figure below and the Pico will extract the information needed and respond accordingly.

Code	Response
201	Activate ToF sensors and start calibration process for IMU.
221	Read the yaw and linear velocity from the IMU.
222	Read and publish all ToF sensor measurements in the form of an array.
322	Localise at the middle of the arena and publish the response.
323	Localise at the start of the arena and publish the response.

Table 3.8.1 - Standard codes used for system cross communication

3.9 System Testing

System testing was conducted with two levels of tests being undertaken. System integration testing was conducted on each feature in isolation running the selected feature in isolation and then measuring its expected output relative to our input. The second level of testing we conducted was system qualification testing, this is where the system was run as a complete unit and all features were checked to ensure that they interfaced with each other and ran as designed.

Parcel Delivery Integration Test

Preconditions:

Step	Action
1	Robot is assembled with the parcel delivery system
2	Robot is placed within 10cm of the destination with parcel drop facing the drop off location
3	Plug the servo power and ground into the raspberry pi and also the PWM signal into GPIO_18
4	Power on the Raspberry Pi

Table 3.8.2 - Preconditions for delivery integration test

Test Steps:

Step	Action	Expected Response
1	Run Test Code	None
2	Wait 1 second whilst code executed parcel hold state	None
3	Place a parcel of size 150x150x150mm (l*w*h)	Parcel is held in the delivery bay
4	Wait 5 seconds whilst code executes parcel delivery	Delivery flap opens and parcel slides into the destination box
5	Wait 2 seconds as code executes flap return	Delivery flap return to loaded position and servo turns off

Table 3.8.3 - Test steps for delivery integration test

Satisfied Requirement IDs:

13, 14

NFC Integration Test

Preconditions:

Step	Action
1	Raspberry Pi is connected to RFID reader as per wiring diagram in section 3.6
2	3 NFC tags are programmed with data variables “1”, “2”, “3”

Table 3.8.4 - Preconditions for NFC integration test

Test Steps:

Step	Action	Expected Response
1	Run Test Code	None
2	Scan First NFC tag	Check command window prints out corresponding data variable eg. “1”
3	Repeat step 2 for remaining 2 NFC tags	None

Table 3.8.5 - Test steps for NFC integration test

Satisfies Requirement ID’s:

11

Navigation Integration Test

Preconditions:

Step	Action
1	Robot parameters are calibrated with wheel radius and robot width.
2	For Real Life test, robot serializer functional with wheel encoders.
3	Goals

4	Plotting variable is set to True.

Table 3.8.6 - Preconditions for navigation integration test

Test Steps:

Step	Action	Expected Response
1	Test A* Algorithm	Plotted Map should detail each waypoint to reach desired goal.
2	Test Tentacle Planner	Plotted Map should show robot driving to waypoints.
3	Test Simulation Code	Plotted Map of robot in a simulated environment should be able to navigate waypoint to waypoint.
4	Test Connection to Arduino Nano (Wheel controllers)	Robot prints serialised strings between arduino and pi, which should print current wheel velocities and send desired wheel velocities.
5	Test Connection to Raspberry Pico (IMU System)	Robot prints serialised strings between arduino and pi, which includes robot's orientation and localization.
6	Test Real Time System	Robot position is accurate (pose, orientation), and able to perform localization to correct position.

Table 3.8.7 - Test steps for navigation integration test

Satisfies Requirement ID's:

4, 9

Communication Integration Test

Preconditions:

Step	Action
1	Bluetooth is enabled on the robot.
2	Both Bluetooth MAC Addresses are entered between the robots in code.

3	Bluetooth between robots successfully paired using “bluetoothctl”

Table 3.8.8 - Preconditions for communications integration test

Test Steps:

Step	Action	Expected Response
1	Connect both robots together using Bluetooth (“bluetoothctl”).	Bluetooth pairing / connection confirmation on other robot in terminal, enter “YES”
2	Run Bluetooth Server / Client python test code	Server established, python code will print sent data and receiving data.

Table 3.8.9 - Test steps for communications integration test

Satisfies Requirement ID’s:

8

Sensor Integration Test

Preconditions:

Step	Action
1	Setting up a 3.3V power rail to power the sensors.
2	Setting up power for the Raspberry Pi Pico.
3	Connect the sensors in the same format as Section 3.3.
4	Download the files “I2CTest.ino” and “multisensor_archi_hq.ino” from the project’s GitHub (Appendix A) repository. The two files can be found in the “Sensor‘test_code/Testing Code Sensor” Folder.
5	Connect the Raspberry Pi Pico to the computer and open up the serial console.

Table 3.8.10 - Preconditions for sensor integration test

Test Steps:

Step	Action	Expected Response

1	Upload the code file “I2CTest.ino” onto the Pico	None
2	Wait for the code to upload	None
3	Check if the correct default addresses of the sensors are printed onto the serial console.	Serial print “I2C device found at: 0x29”
4	Upload the code file “multisensor_archi_hq.into” onto the Pico.	None
5	Wait for the code to upload and sensors to set up.	None
6	If all things are running correctly, continuous sensor readings should be printed out.	Serial print “xxx, xxx, xxx ,xxx …, xxx”

Table 3.8.11 - Test steps for sensor integration test

Satisfies Requirement ID’s:

Satisfies System Level Requirement Only not User Requirement

Microcontroller Communication Integration Test

Preconditions:

Step	Action
1	Setting up a 3.3V power rail to power the sensors.
2	Setting up power for the Raspberry Pi Pico.
3	Connect the sensors as shown in Section 3.3.
4	All the sensors are working and tested (Sensor Integration Test has been completed).
5	Open the project’s GitHub page (Appendix A), download the file “Pico-10DOF-IMU_Rev2_1.ino” from the “arduino/Pico-10DOF-IMU_Rev2_1” folder.

6	Open the project's GitHub page (Appendix A), download the file "test_env.ipynb" and "test_env.py" from the "test_code/Testing Code Sensor" folder.
7	Connect the Raspberry Pi Pico to the computer and open up the serial console.

Table 3.8.12 - Precondition for microcontroller communication integration test

Test Steps:

Step	Action	Expected Response
1	Upload the code file "Pico-10DOF-IMU_Rev2_1.ino" onto the Pico	None
2	Wait for the code to upload	None
3	Run all the code chunks inside the "test_env.ipynb" file	Corresponding responses described in 3.8.3.2.

Table 3.8.13 - Test steps for microcontroller communication integration test

Satisfies Requirement ID's:

Satisfies System Level Requirement Only not User Requirement

Battery Safety Test

Preconditions:

Step	Action
1	Robot electrical system is assembled
2	1 Jumper Wire is placed into the positive input to the buck converter

Table 3.8.14 - Preconditions for battery safety test

Test Steps:

Step	Action	Expected Response
1	Power on Robot	None

2	Short jumper wire into ground	Inline battery fuse blows

Table 3.8.15 - Test steps for battery safety test

Satisfies Requirement ID's:

19

Single Robot Qualification Test

Preconditions:

Step	Action
1	Whole Robot is Assembled and Powered
2	Calibrate Robot
3	Robot is placed in the start zone

Table 3.8.16 - Preconditions for single robot qualification test

Test Steps:

Step	Action	Expected Response
1	Start execution of full system test code and start 5 minute time	None
2	Load parcel and scan corresponding NFC tag	Robot Begins to Move to location after 2 seconds
3	Observe robot for smooth movement	Robot travels smoothly to the centre of the arena then to the scanned destination. Robot stops at the destination and drops off the parcel. Robot returns to the centre of the arena then localises. Robot returns to the loading zone.
4	Repeat steps 2-3 until 5 minute timer elapses	

Table 3.8.17 - Test steps for single robot qualification test

Satisfies Requirement ID's:

4, 7, 9, 11, 13, 14, 17

Two Robot Qualification Test

Preconditions:

Step	Action
1	Bluetooth systems are setup (See Bluetooth Integration Test)
2	State Machine setup with appropriate wait states and set states.

Table 3.8.18 - Preconditions for two robot qualification test

Test Steps:

Step	Action	Expected Response
1	Parallel State Test using isolated state machine test (uses sleep function for timing)	Prints out states of both robots which should follow the state machine system detailed in Section 3.7.
2	Run Full System Test using a simulated robot	Robot would follow the state machine system detailed in Section 3.7 as well as executing commands (navigating, delivery, localization)
3	Full Test between two robots.	Both robots would follow the state machine system detailed in Section 3.7, as well as executing commands (navigating, delivery, localization)

Table 3.8.19 - Test steps for two robot qualification test

Satisfies Requirement ID's:

4, 7, 8, 9, 11, 13, 14, 17

3.10 Revised Bill of Materials

The following section details the bill of materials used for the project. Detailed cost breakdown comparison with the proposal is covered in Section 3.11.

Item	Quantity	Unit Cost (AUD)	Cost (AUD)	Source Idx
Motors with Encoders	2	$73	$146	1
Raspberry Pi 4, Model B (4GB)	1	$94	$94	2
Arduino Nano Microcontroller	1	$3.85	$3.85	3
Raspberry Pi Pico Microcontroller	1	$6.30	$6.30	4
Buck Converter	1	$19.95	$19.95	5
Vl53l0x ToF Sensor	6	$2.63	$15.75	6
Vl53l1x ToF Sensor	2	$2.15	$4.30	7
MPU9250	1	$3.68	$3.68	8
Sg90 Mini Servo	1	$0.79	$0.79	9
Lipo Battery	1	$33	$33	10
DFRobot 2x1.2A DC Motor Driver	1	$7.02	$7.02	11
RFID sensor	1	$2.49	$2.49	12
Total Cost			$337.13	

Table 3.10.1 - Revised bill of materials

Index	Source Website
1	https://www.pololu.com/product/4866

2	https://au.element14.com/raspberry-pi/rpi4-modbp-4gb/raspberry-pi-4-model-b-4gb/dp/3051887?gclid=CjwKCAjw7c2pBhAZEiwA88pOF3bHsdbAIvi3moxgWw7vrOD7USiWv7FHifGe3A7klrEef_EyQvmOzBoCxGYQAvD_BwE&mckv=_dc|pcrid||pkw||pmt||slid||product|3051887|pgrid||ptaid||&CMP=KNC-GAU-GEN-SHOPPING-PLA-PMAX
3	https://www.aliexpress.com/item/1005006102453114.html?src=google&aff_fcid=a1dd84c8603047f69537c8c44cc4e832-1697952951895-03511-UneMJZVf&aff_fsk=UneMJZVf&aff_platform=aaf&sk=UneMJZVf&aff_trace_key=a1dd84c8603047f69537c8c44cc4e832-1697952951895-03511-UneMJZVf&terminal_id=b42d72490e60489e9a22a952b4e9af27&afSmartRedirect=y
4	https://core-electronics.com.au/raspberry-pi-pico.html
5	https://www.amazon.com.au/1PZ-Converter-Regulator-24V%EF%BC%887-36V-1-25-32V%EF%BC%89/dp/B097LDNTBY/ref=d_pd_vtp_sccl_3_3/358-4804098-7408819?pd_rd_w=ihiSD&content-id=amzn1.sym.29c7528a-1f65-4284-ac6a-187bc2d1984c&pf_rd_p=29c7528a-1f65-4284-ac6a-187bc2d1984c&pf_rd_r=EHSCMVJMAHAAE56X1YSN&pd_rd_wg=ALc0H&pd_rd_r=aa8ae578-7f7a-45c2-8b06-df102e369681&pd_rd_i=B097LDNTBY&psc=1
6	https://www.aliexpress.com/item/1005005669082817.html?src=google&aff_fcid=a148f6854f7049ef9e97a17a88f4ce6f-1697953328785-02759-UneMJZVf&aff_fsk=UneMJZVf&aff_platform=aaf&sk=UneMJZVf&aff_trace_key=a148f6854f7049ef9e97a17a88f4ce6f-1697953328785-02759-UneMJZVf&terminal_id=b42d72490e60489e9a22a952b4e9af27&afSmartRedirect=y
7	https://www.aliexpress.com/item/4000116535518.html?src=google&aff_fcid=0979b7abd7b44d7bb70554aa2aa2966c-1697953424033-04086-UneMJZVf&aff_fsk=UneMJZVf&aff_platform=aaf&sk=UneMJZVf&aff_trace_key=0979b7abd7b44d7bb70554aa2aa2966c-1697953424033-04086-UneMJZVf&terminal_id=b42d72490e60489e9a22a952b4e9af27&afSmartRedirect=y
8	https://www.aliexpress.com/item/1005006047556808.html?src=google&src=google&albch=shopping&acnt=631-313-3945&slnk=&plac=&mtctp=&albbt=Google_7_shopping&albagn=888888&isSmbActive=false&isSmbAutoCall=false&needSmbHouyi=false&albcp=19812794076&albag=&trgt=&crea=en1005006047556808&netw=x&device=c&albpg=&albpd=en1005006047556808&gclid=CjwKCAjw7c2pBhAZEiwA88pOF7arc-jriO3lb3HvlxDXi7eizKVMZER4b2PA9qmiWEd1hSK1sWscLhoCwAIQAvD_BwE&gclsrc=aw.ds&aff_fcid=4356a0e4a7d846b8b95b451d306d3eae-1697953482227-04149-UneMJZVf&aff_fsk=UneMJZVf&aff_platform=aaf&sk=UneMJZVf&aff_trace_key=4356a0e4a7d846b8b95b451d306d3eae-1697953482227-04149-UneMJZVf&terminal_id=b42d72490e60489e9a22a952b4e9af27&afSmartRedirect=y
9	https://www.aliexpress.com/item/1005005929037037.html?spm=a2g0o.productlist.main.3.2acef47bwAz2hs&algo_pvid=b4ce4566-1b5f-4036-b65f-ffea4a73165a&aem_p4p_detail=20231021224638178157327231700024235 86&algo_exp_id=b4ce4566-1b5f-4036-b65f-ffea4a73165a-1&pdp_npi=4%40dis%21AUD%210.98%210.79%21%21%210.61%21%21%402101fb1016979535980866882e3240%2112000003489072513 2%21sea%21AU%213875491488%21&curPageLogUid=p1rZ7z339Zkf&search_p4p_id=20231021224638178157327231700024235 86_2
10	https://www.amazon.com.au/Zeee-Battery-5200mAh-Vehicles-Airplane/dp/B09NXCG8VC/ref=sr_1_1_sspa?adgrpid=85110744937&hvadid=591310160066&hvdev=c&hvlocphy=9071364&hvnetw=g&hvqmt=e&hvrand=16851247058633162889&hvtargid=kwd-420940293004&hydadcr=13547_360595&keywords=amazon+lipo+battery&qid=1697953643&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1
11	https://au.element14.com/dfrobot/dri0044/dc-motor-driver-2-7v-to-5-5v-2ch/dp/3974106?gclid=CjwKCAjw7c2pBhAZEiwA88pOF4koe47aj4otUyzzlXHredROXoNhW6Y-dgP-8lJV1ldX7D0VsOQNbhoCFXMQAvD_BwE&mckv=_dc|pcrid||pkw||pmt||slid||product|3974106|pgrid||ptaid||&CMP=KNC-GAU-GEN-SHOPPING-PLA-PMAX
12	https://www.aliexpress.com/item/1005006028041323.html?src=google&aff_fcid=f3f7c32abb574f648b4d0b654801cbcd-1697953821244-05181-UneMJZVf&aff_fsk=UneMJZVf&aff_platform=aaf&sk=UneMJZVf&aff_trace_key=f3f7c32abb574f648b4d0b654801cbcd-1697953821244-05181-UneMJZVf&terminal_id=b42d72490e60489e9a22a952b4e9af27&afSmartRedirect=y

Table 3.10.2 - Price sources for revised bill of materials

3.11 Design Comparison

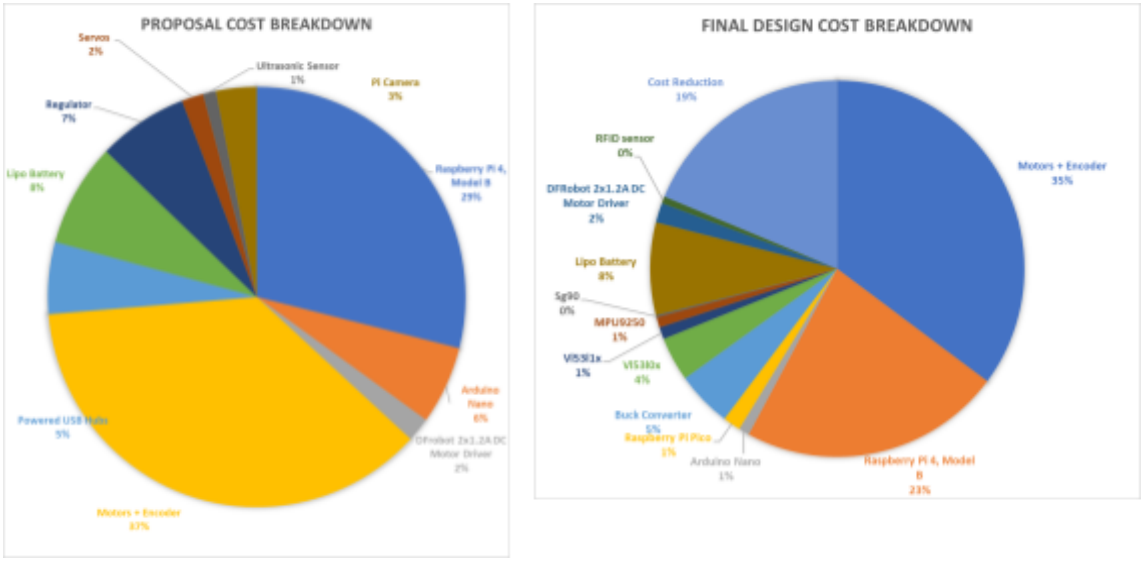


Figure 3.11.1 - Cost breakdown of the robot. (Left) Proposal Cost Breakdown (Right) Final Design Cost Breakdown.

Estimated Budget	Final Budget
$414.81	$337.13

Table 3.11.1 - Budget estimation vs final budget

A 19% cost reduction can be observed when comparing the estimated budget to the final budget. This is a result of some key design changes, market factors and changes in the component sourcing strategies.

Change Made	Explanation	Result
Design Changes		
Change from camera based package recognition to RFID based recognition	- Originally the proposed system hopes to utilise camera systems for the package recognition and homographies. In the end however, due to the difficulty and instability of such a system (highly sensitive to lighting of the environment), the cheaper and more robust option of RFID based recognition is selected.	Cost reduction

Removal of Power USB Hub	- The powered usb-hub is no longer utilised in the design, extra effort is put into creating two power rails for supplying power to the sensors.	Cost reduction
Change of Sensors from Ultrasonic to Tof	- Sensors are changed from Ultrasonic sensors to ToF sensors. - The advantages and trade-offs of this change is discussed in depth in Section 3 of Design Overview	Cost Increase
Adding IMU to the system	- IMU is added to the system to improve the estimation of the robot's yaw angle.	Cost Increase
Changes in the Sourcing Strategies		
Pico and Nano	- The Pico and Nano's price vary dramatically depending on the source. - Arduino Nano being an old, open sourced board, the board's quality does not vary significantly from one supplier to another. - Pico price can be cut even more if the buyer is able to solder male connect headers onto the microcontroller.	Cost reduction
Ali-express	- Ali-express is often viewed as an unreliable source for electronic components due to its mixed quality, lack of documentation, hidden prices and unreliable shipping time. - However after examining the source closely it is realised that a lot of the components' retailers sell in Australia, are sourced from Chinese manufacturers, many of which are directly contactable through Ali-express. - Therefore during the 2nd phase of the project, Ali-express is selected as a potential source of components supply and some of	Cost reduction

	the components are sourced from Ali-express.	
Market Factors		
Raspberry Pi Cost reduction	- Raspberry Pi has recently pushed out its newest Raspberry Pi 5 onto the market[17]. - Pi 5 has a significant improvement in performance compared to Pi 4, leading to a lot of the suppliers cutting down the price of Pi 4, in face of the new product.	Cost reduction

Table 3.11.2 - Reasons for cost changes

4.0 Competition Strategies

The robot will perform a series of basic actions in order to perform the tasks of parcel delivery. These tasks together will constitute an entire workflow, and consists of:

1. Load the parcel
2. Recognise the parcel
3. Plan a path towards the destination of the parcel
4. Drive to the destination
5. Stop at the destination
6. Unload the parcel
7. Plan a path towards the loading zone
8. Drive to the loading zone
9. Await further instructions

Each individual robot will be responsible for its own parcel loading, path finding, and delivery. When seen in isolation, the robot will deliver the parcel, return to the loading zone, and localise at the loading zone to correct for any drift that may have accumulated during the robot's delivery.

The robot will localise itself by parking itself a certain distance from a corner in the loading zone, and ascertain its location with sensors by measuring its distance from both walls of the corner. The robot will also localise itself at the middle of the arena, during its delivery runs. It will do this by using distance sensors on all sides of the robot, to ascertain the distance between itself and all four walls of the arena. An illustration can be seen in figure 4.0.1.

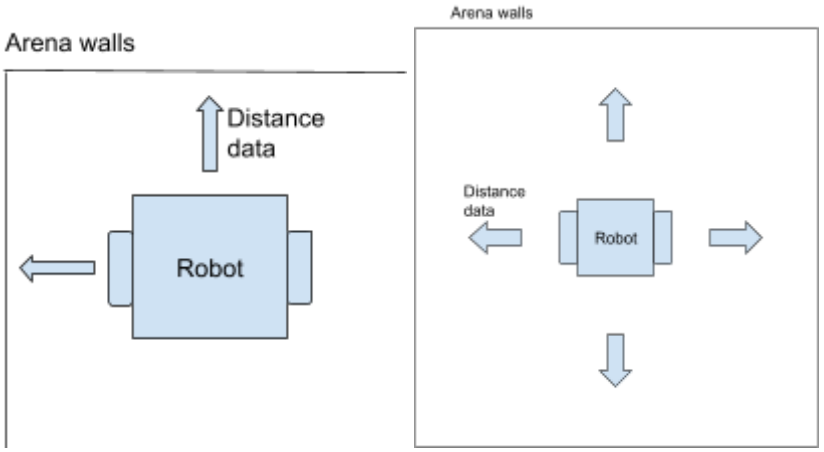


Figure 4.0.1 - the robot ascertains its position by measuring its distance from two wall in the loading zone (left), and from all four wall when located at middle of arena (right)

In order to collaborate with other robots operating in the same arena, the robot has been programmed to not interfere with the other robots. A common communication standard has

been devised and set up between the two robots. This standard is communicated through bluetooth, and allows the robot to report itself in the following states:

1. Initialisation
2. Robot is loaded and ready
3. Robot is in delivery zone
4. Robot has returned from delivery zone
5. Robot is loading parcels in loading zone

During the competition, the two robots will operate with one robot lagging the other robot by two states. That is, for example, while one robot is in state 3, ie, currently operating in the delivery zone, the other robot must be in state 1, ie, initialising for the next run. This ensures both robots are performing different tasks at any given time, and are physically separated from each other to minimise the risk of interference between the two robots and thus eliminates the need for collision avoidance.

5.0 Project Management Retrospective

The team was able to adapt to design changes, and were able to manage agile development of the robot. We were able to restructure our project plan successfully when the design decision was made to switch from ultrasonic sensors to time of flight sensors and an inertial measurement unit. Strong communication allowed us to have each team member work on their assigned tasks in parallel given the short redesign timeframe.

The plan of the base tasks such as robot assembly occurring before other integration tasks allowed a swift transition of the robot between team members, allowing them to implement their design tasks directly to the robot. A key example would be that the robot assembly was completed and then the robot was handed over to allow sensor integration in the code, afterwards it was finally handed to the navigation team which relied on everyone's prior work to be completed. Another key strength was the communication between the feature teams, feature teams that were reliant upon each other were always up to date with each other's implementation strategy allowing for seamless integration of all features.

A key shortcoming of our planning occurred during the lead up to milestone 1. As we had delegated features based on milestone 2 we had team members who were not working on tasks relevant to the first milestone. Tasks such as the 3D design of the parcel delivery should have been delayed, to allow for team members to assist on tasks that are more immediate. A heavy burden was placed on Joel who was developing navigation for milestone 1, and this could have been avoided by re-assigning a team member who did not have immediate deadlines for milestone 1 to provide support.

5.1 Team Roles

Terence

Responsible for the camera pose estimation section of the localisation, as well as assisting in calibrating sensors and navigation. As Terence had access to a laptop with a Graphics Processing Unit, homography simulations could be run quickly and without much CPU usage, allowing for more testing and fine tuning of pre-processing images to improve the homography.

Yide

Responsible for implementing the micro-controllers, sensor system and driver module. Yide has a large repository of existing components and has some experience with basic sensors, therefore he is assigned the tasks of selecting the right sensor to use for the project, leading the sensor implementation and helping interface these components to the master controller (Pi-4).

Luqi

Responsible for the integration of initially the ultrasonic sensors into the robot, and later was responsible for integrating and preparing the RFID parcel recognition system into the final robot. Luqi was also responsible for creating CAD models and concept designs for the mechanical components of the robot.

Joel

Responsible for navigation systems and robot communication protocols, and acting as the main python software engineer for navigating the github. Later, Joel worked on integration of the robot systems for the competition.

Ayrton

Ayrton was responsible for all of the mechanical assembly and electrical assembly. As Ayrton had access to 3D printers and tools he was able to quickly prototype parts to ensure that they fit the robot. As Ayrton also had access to an electronics lab through his work he was able to solder and wire up all the interconnections on the robot, and subsequently test them to ensure there were no shorts and continuity throughout the circuits.

6.0 Recommendations for Future Work

6.1 Cost reduction

A major issue that the team encountered during the design and construction phases of the project is that the cost of the proposed robot design vastly exceeded the project limit. As one of the initial design requirements, the team was tasked to build the robot with a $200 limit on the cost. Although the team made efforts to reduce costs as compared to the initial proposal, the $200 limit was not met by the team.

In order to produce a more capable robot, with more processing power to handle complex tasks, the team decided to use multiple different mirco-processors and computing units, such as Arduino and Raspberry Pi, in parallel with each other. Furthermore, the provided motors also added significant cost to the project. While this did allow the robot to become more capable, it also meant that significant cost was expended on acquiring these units.

In retrospect, the robot may have been able to satisfy all the user requirements while not having to use more than what is originally provided, that is, a sole Raspberry Pi, perhaps in combination with a PSoC board. Therefore, it is recommended for future work to simplify the robot by using fewer processors. Although such actions may reduce performance, it will be able to cut down on costs as well, and produce a simpler, cheaper robot.

6.2 Modifications to cooperation protocols

It was noticed that during the competition, having both robots perform their individual path finding, whilst one robot maintaining a lag behind the other by two states (see section 4, Competition Strategies for detail), proved to be very inefficient. The path finding process was slow, and such slowness would also impact the other robot within the arena, by forcing the other robot to sit idle whilst the path finding of the first robot is being performed.

Thus, it is recommended that for future work, a modification of protocols for the two robots cooperating, and also pathfinding through the arena to be modified. It would be most ideal that the two robots remain physically separated in a way similar to the current protocol, whilst the protocol should also prevent any down time in the operation by allowing both robots to be in constant motion.

A proposed protocol is that both robots should follow a clockwise wall following algorithm, with one robot trailing the other robot by 180 degrees. An illustration of the algorithm can be seen in figure 6.2.1.

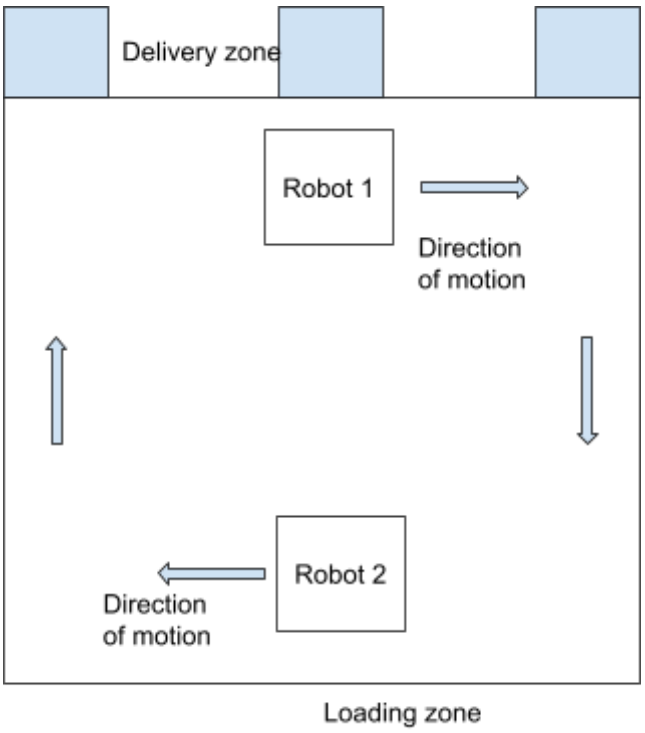


Figure 6.2.1 - Illustration of the wall following algorithm.

This algorithm would physically separate the two robots by having them operate at the opposite sides of the arena to one another, whilst keeping both robots in constant motion. By having the robots following the walls, it prevents the robot from cutting through the middle of the arena, and potentially get in the way of the other robot, which would force one of the robots to stop and wait for the other to pass. This would increase the efficiency of the parcel delivery process, and would increase the number of parcels delivered in a set amount of time.

7.0 Conclusion

Overall, the developed robot was able to meet all user requirements. The design process of the robot proved to be turbulent, as several aspects of the robot had to be adapted significantly as the project progressed, due to various factors. Most significant of which included the sensors, which the team decided on ToF sensors, given how other sensors were either too unreliable or too expensive. The testing process of the robot raised several issues that were in need of improvement, such as how the servo controlling the parcel gate had to be constantly locked to prevent unwanted openings.

The final competition was successful, and the robot was able to complete the tasks required of it, by successfully recognising and delivering a number of parcels in a span of five minutes. It can be said that the development progress of the robot had some great difficulties, such as the integration process of all the subsystems, and certain areas of the robot's performance can be improved, such as delivery efficiency, and cost. The final robot is nevertheless a capable robot, and able to successfully perform all tasks the customer requires of it.

Bibliography

[1] M. Burke, “Project Brief.” Accessed: Oct. 22, 2023. [Online]. Available: https://docs.google.com/document/u/1/d/12684r1Q4GQoWifl5HNDJErfUojEXFgcmdmOsXUJHWnI/edit?usp=sharing&usp=embed_facebook

[2] R. P. Ltd, “Raspberry Pi 4 Model B specifications,” Raspberry Pi. Accessed: Oct. 22, 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[3] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah, and E. Gunawan, “A Systematic Literature Review of A* Pathfinding,” *Procedia Comput. Sci.*, vol. 179, pp. 507–514, Jan. 2021, doi: 10.1016/j.procs.2021.01.034.

[4] “socket — Low-level networking interface,” Python documentation. Accessed: Oct. 22, 2023. [Online]. Available: https://docs.python.org/3/library/socket.html

[5] “Encoder: channels A and B, phase shift and direction of rotation,” maxon Support. Accessed: Oct. 22, 2023. [Online]. Available: https://support.maxongroup.com/hc/en-us/articles/360005962273-Encoder-channels-A-and-B-phase-shift-and-direction-of-rotation

[6] “MPU-9250 Datasheet | TDK InvenSense.” Accessed: Oct. 22, 2023. [Online]. Available: https://invensense.tdk.com/download-pdf/mpu-9250-datasheet/

[7] E. F. P. III, “Arduino-Pico.” Oct. 20, 2023. Accessed: Oct. 22, 2023. [Online]. Available: https://github.com/earlephilhower/arduino-pico

[8] *Arduino VS CircuitPython Speed Comparison*, (Dec. 20, 2022). Accessed: Oct. 22, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=qym-P4GTdIU

[9] S. O. H. Madgwick, “An efficient orientation filter for inertial and inertial/magnetic sensor arrays”.

[10] S. A. Ludwig, K. D. Burnham, A. R. Jiménez, and P. A. Touma, “Comparison of attitude and heading reference systems using foot mounted MIMU sensor data: basic, Madgwick, and Mahony,” vol. 10598, p. 105982L, Mar. 2018, doi: 10.1117/12.2296568.

[11] “Yaw/Heading optimization by drift elimination on MEMS gyroscope - ScienceDirect.” Accessed: Oct. 22, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0924424721001540?fr=RR-2&ref=pdf_download&rr=819eee1cdb8829ab

[12] “A* Search Algorithm,” GeeksforGeeks. Accessed: Oct. 22, 2023. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/

[13] “AtsushiSakai/PythonRobotics: Python sample codes for robotics algorithms.” Accessed: Oct. 22, 2023. [Online]. Available: https://github.com/AtsushiSakai/PythonRobotics/tree/master

[14] “Evidential-Based Approach for Trajectory Planning With Tentacles, for Autonomous Vehicles | IEEE Journals & Magazine | IEEE Xplore.” Accessed: Oct. 23, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/8788688

[15] “mfrc522.” Pi My Life Up, Oct. 16, 2023. Accessed: Oct. 23, 2023. [Online]. Available: https://github.com/pimylifeup/MFRC522-python

[16] “I2C - SparkFun Learn.” Accessed: Oct. 23, 2023. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c/all

[17] R. P. Ltd, “Raspberry Pi 5,” Raspberry Pi. Accessed: Oct. 23, 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/

Appendix

Appendix A: Github Code Repository

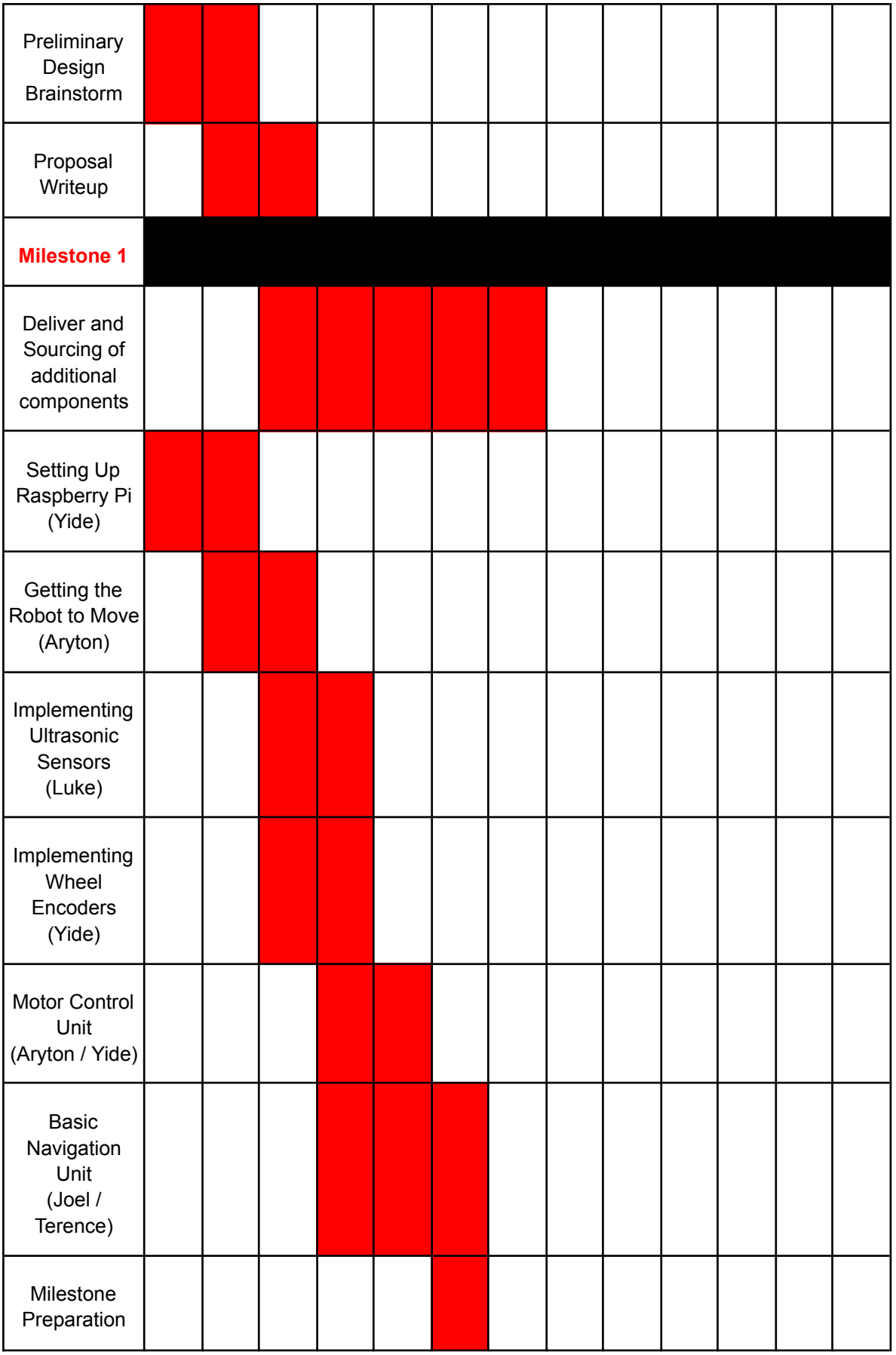
The link to the group's GitHub Code Repository:
https://github.com/jong0052/ECE4191Group1/tree/main

Appendix B: Project Timeline

The following describes the project timelines for the duration of the project.

Table 1: Actual Project Timeline

	Week												
Goals	1	2	3	4	5	6	7	8	9	10	Break	11	12
Key Deadlines													
Project Proposal													
Milestone 1 - Demonstrations - Individual Videos (1 minute)													
Milestone 2 - Demonstrations - Group Video (5 min) - Final Report - Design Evaluation													
Project Proposal													
Roles Allocation													


Preliminary Design Brainstorm
Proposal Writeup
Milestone 1
Deliver and Sourcing of additional components
Setting Up Raspberry Pi (Yide)
Getting the Robot to Move (Aryton)
Implementing Ultrasonic Sensors (Luke)
Implementing Wheel Encoders (Yide)
Motor Control Unit (Aryton / Yide)
Basic Navigation Unit (Joel / Terence)
Milestone Preparation

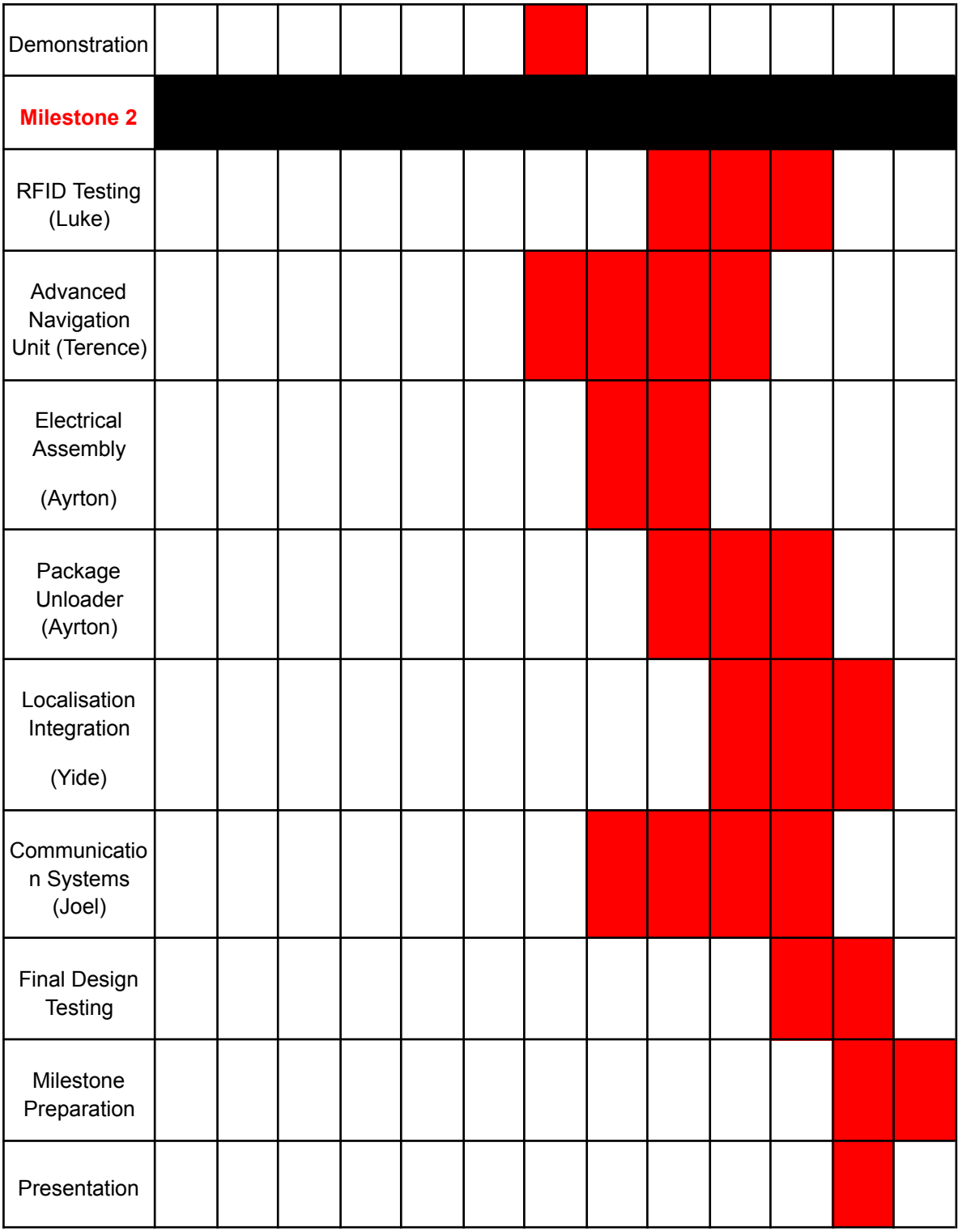
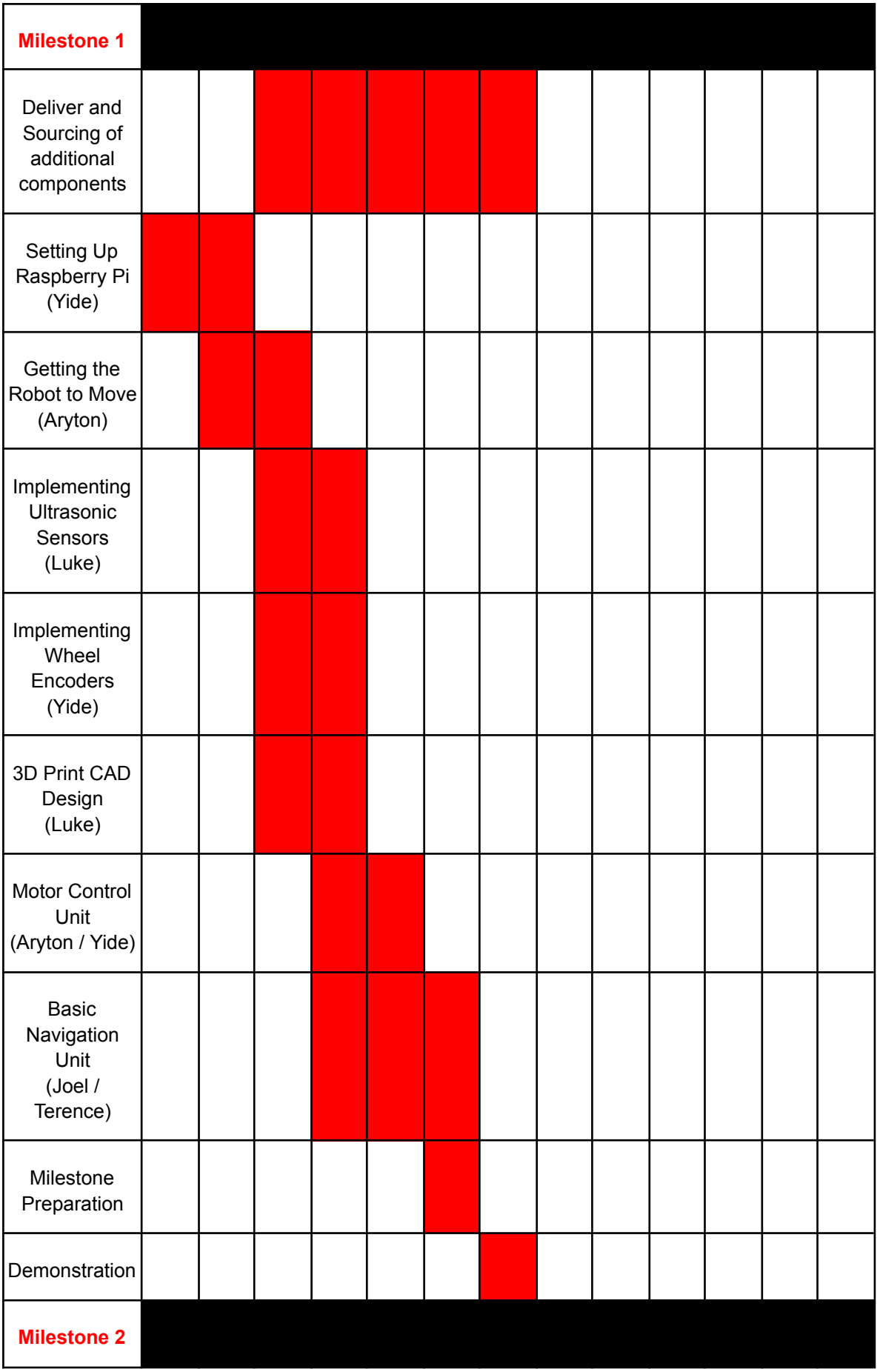

Demonstration
Milestone 2
RFID Testing (Luke)
Advanced Navigation Unit (Terence)
Electrical Assembly (Ayrton)
Package Unloader (Ayrton)
Localisation Integration (Yide)
Communicatio n Systems (Joel)
Final Design Testing
Milestone Preparation
Presentation

Table 2: Proposal Timeline

	Week												
Goals	1	2	3	4	5	6	7	8	9	10	Break	11	12
Key Deadlines													
Project Proposal													
Milestone 1 - Demonstrations - Individual Videos (1 minute)													
Milestone 2 - Demonstrations - Group Video (5 min) - Final Report - Design Evaluation													
Project Proposal													
Roles Allocation													
Preliminary Design Brainstorm													
Proposal Writeup													


Milestone 1
Deliver and Sourcing of additional components
Setting Up Raspberry Pi (Yide)
Getting the Robot to Move (Aryton)
Implementing Ultrasonic Sensors (Luke)
Implementing Wheel Encoders (Yide)
3D Print CAD Design (Luke)
Motor Control Unit (Aryton / Yide)
Basic Navigation Unit (Joel / Terence)
Milestone Preparation
Demonstration
Milestone 2

RFID Testing (Aryton)													
Package Loader (Yide)													
Advanced Navigation Unit (Terence)													
Package Unloader (Luke)													
Communicatio n Systems (Joel)													
Final Design Testing													
Milestone Preparation													
Presentation													